Passau University

Bachelor Thesis

Generating Urban-like Scenarios to Spot Fuel-inefficient Behaviour of Autonomous Cars

Author:
Michael Heine

Supervisor:
Prof. Dr.-Ing. Gordon Fraser

Advisor:
Alessio Gambi, Ph.D.

Monday 28th September, 2020

Abstract

Researchers proposed many groundbreaking ideas and techniques to make autonomous vehicles safer and more reliable on the streets. But society still has many concerns about the safety and reliability of autonomous cars. Many contributions focused on the most important property: the safety of passengers. But also other non-functional properties have to be ensured. One of them is the fuel consumption of cars. Nobody wants to drive in a car which consumes fuel and pollutes the environment more than necessary. My thesis aims to expose the problems of autonomous cars concerning their fuel-inefficiency. By defining a set of scoring oracles, the decisions taken by a car can be assessed. These oracles check several sensor values and driving patterns to classify whether the car under test drives fuel-inefficient or not, e.g. the ego-car drives with a high RPM (rounds per minute) with a low gear and hence, this is an infraction against the oracle. Every infraction is logged and then used to calculate a scoring function. By using procedural content generation, a method commonly used in the gaming industry to algorithmically generate various content, I randomly create urban-like scenarios with intersections, traffic, traffic lights and signs as well as parked cars to stress fuel-inefficient driving behaviour of self-driving vehicles. The evaluation results show that my scoring function can expose faults concerning fuel-inefficiency of different driving behaviours. I also proved a strong positive correlation of 0.779 between the score and consumed fuel. My test cases, on the other hand, exposed many faults of a traffic light detection system.

Contents

1 Introduction and Motivation

According to the United States Environmental Protection Agency, the transportation sector in the US produces 28% of the total greenhouse gas emissions in the country [19]. There are many ways to reduce overall emissions. One would be using public transportation, but since not everyone has enough money to purchase a ticket or no opportunity to use one because of geographical reasons, this method mainly holds for people who live in higher populated cities. Another aspect would be buying cars with electric engines, but the prices are generally higher compared to petrol-powered engines. Hence, the market share of e-vehicles is smaller. Also, there are only limited places where a holder could charge his or her car. Another idea would be sharing the car with other people, also known as carpooling. The problem is that many people have low trust in strangers or picking other passengers up would cause the driver to drive a long detour. Researchers try to find alternatives for petrol. For example, Liaquat et al. [48] propose coconut biodiesel to reduce gas emissions. The problem is that many proposed alternatives couldn't make their way to the public interest to be sold at gas stations.
Right now, one of the best possibilities to reduce fuel consumption is to adopt a fuel-efficient drive style. For example, accelerating the vehicle aggressively after a complete stop increases the fuel consumption up to 40% [17]. According to Morgan Stanley Research [59], if drivers could adopt a more fuel-efficient driving style in the US, this would save up to $158 billion annually and strongly reduce the emission of greenhouse gases. Driving schools teach their students to adopt such driving behaviour by giving them instructions which they learn to obey. However, many drivers seem to ignore or forget these steps and therefore, there must be more initiatives to remind people how to drive fuel-efficient.
Car manufacturers already added features to decrease fuel consumption. Many modern vehicles have built-in cruise-control to keep a constant speed which is very effective on highways. Nissan [51], for example, increased the resistance of the gas pedal after a certain threshold so people would learn to accelerate slowly. These tools are helpful to decrease fuel consumption, but the major factor is still the driving behaviour of people.
While autonomous cars keep evolving and improving, they can overcome this issue in the future. Their goal is to drive better than humans to reduce or completely avoid accidents. At the moment, most of the research efforts in this domain currently focus on ensuring the safety of passengers and other traffic participants. Other non-functional properties like passenger comfort or fuel-efficiency are mostly neglected at the moment. Because self-driving cars are programmed, their behaviour can be validated and verified in simulations before deploying them on real roads, which allows to optimize and fine-tune them. That's why autonomous cars have the potential to reduce the overall fuel consumption by adopting a smooth and fuel-efficient driving behaviour. This is where test cases checking whether self-driving cars are truly fuel-efficient or not become very useful. Moreover, improving the fuel-efficiency of cars should not affect other properties like safety and passenger comfort in any bad ways. Having an economy-friendly driving behaviour does not only save fuel but can also increase the driver's comfort due to smooth driving and reduce costs. These are some requirements that lead society to accept autonomous cars [51].
Before self-driving cars can be deployed in the real world, they must be extensively tested to ensure that they meet all important criteria like obeying traffic rules and ensuring passenger comfort and safety. Otherwise, failures can be costly for manufacturers and even more important might cause deadly accidents like the Tesla car crash [27] or the Uber car [49]. There are two popular approaches to test self-driving cars: in the real world and via simulation. Testing self-driving car software using simulations is cheaper and faster than letting the car drive in the real world. Also, a programmer can create infinite test cases while test environments, in reality, are strictly limited and not every scenario can be created or verified. Additionally, simulations

nowadays can be very accurate concerning driving physics and new content can be easily added, motivating their use as a crucial quality assurance check during development.
For this thesis, I identified some critical driving scenarios in urban environments to assess fuel-inefficient driving behaviour of autonomous cars. Those are intersections and roads with varying curvature and length. Both allow assessing fuel-inefficiency in different ways. I implemented a prototype test generator that can generate the necessary urban-like scenarios to implement effectively those relevant driving scenarios and used it to test state-of-art implementations of the software components currently employed in self-driving cars. Urban-like scenarios contain many elements which are impossible to implement within a short time. I focused on some of these elements, which are intersections, traffic lights/signs, traffic participants and parked cars.
By following expert suggestions [51, 18, 42] and common sense, I created a set of "Fuel-specific" test oracles which can be applied to the driving scenarios that I generate. These oracles are driving at high rounds per minute (RPM) value, pressing the throttle or brake too hard, not turning the engine off at intersections and accelerate-and-stop behaviour. My fuel-specific test oracles constantly monitor the ego-car during the tests and log every infraction to define a scoring function. My work, in the spirit, follows the approach of the scoring oracle for the CARLA Challenge [66]. Each infraction increases the score, the penalty is dependent on how bad the infraction was, e.g. driving at 3000 RPM is not as bad as driving at 6000 RPM. My function summarizes the quality of the overall behaviour of the ego-car. One example to stress the ego-car to violate the oracles is to frequently force it to slow down at sharp turns and letting it to come to a complete halt at intersections. That way, the ego-car must re-accelerate every time to reach the target speed, allowing to assess how fuel-inefficient the test subject drives.
To evaluate my approach, I used the realistic vehicle simulator BeamNG [5] with its integrated self-driving AI. I modified the AI's behaviour and created three different driving profiles to see how well several driving styles performed. My scoring function was able to expose the problems of each one and show the differences between them, which matched with the expectations. Hence, my scoring function can reveal fuel-inefficient driving behaviour of any self-driving car. I also proved a strong positive correlation of 0.779 between the consumed fuel and the score, showing that my scoring function can reliably assess the driving behaviour of self-driving cars in terms of their fuel-efficiency. In the third experiment, I deployed a pre-trained traffic light detection system to measure the effectiveness of my test cases with traffic lights. I could expose many problems of the AI, which proves that my test cases are effective in terms of traffic light implementation.
With my work, self-driving car developers can test whether their software drives fuel-efficiently or not. For example, the results revealed that the autonomous car keeps braking too hard at intersections and therefore, there might be problems with the training set or a function calculating the distance to the intersection. Additionally, the scoring oracle can be also used to reveal the fuel-inefficient driving style of human drivers, making them consider to change their driving behaviour. Another use case could be the training of a reinforcement learning agent which eventually will learn how to properly drive the car. My test generator can be modified and used for other purposes, e.g. simulating traffic flow.
It is hard to think about every single specification during development, so my work can help testers to find missing components, e.g. a cruise control system. I hope that my contribution leads self-driving car developers to put greater focus on fuel-efficient driving behaviour.

1.1 Problem Statement

My thesis investigates techniques to expose as many infractions concerning the fuel-efficiency of self-driving cars as possible. The first problem of this was to find formal specifications and expert suggestions to determine how wrong and right actions can be defined. Since there are many sources online, they might have conflicting opinions or instructions, which can't be tested. Also, looking at every existing property which affects fuel consumption is impossible due to time constraints. I considered the most important ones in my opinion, which are explained in Section 3.2. Another problem is how well each property should be approximated to reality. For example, is it enough to consider a gear shift at a specific RPM value or should be the engine load and torque as well as the steepness of the road considered? In my thesis, I focused on the main indicators of each property and made simplifying assumptions for each one.
Another goal that I tried to achieve is to generate test cases that stress specifically fuel-inefficient behaviour. A big challenge was to find a suitable scoring function which is well balanced and fair in terms of penalty points per infraction. The function must correlate on one hand with the consumed fuel and on the other hand with the driving behaviour of the test subject. Furthermore, the scoring oracles must be computable by the data obtained from the simulator.
The most difficult and time-consuming part was implementing the test generator. One aspect was to define how intersections should be generated and connected. This comes also with the challenge of how to place parked cars, traffic lights and signs as well as deploying traffic. Additionally, the test cases should stress the fuel-inefficient driving behaviour of the car under test as much as possible. Another problem was traffic signs and lights placement. They should never stand on the road and be as near to the intersection as possible. Furthermore, the way traffic lights should be controlled was a tricky task because there are numerous possibilities. I had to address questions like when to switch the lights, how long does the current state remain, what should be the initial colour or what kinds of traffic lights to implement. The most difficult task was to create valid test cases with this big amount of features.
The test oracle validates whether the test subjects are violating the traffic rules or not. For traffic lights, the oracle must always know the states of the traffic lights. For traffic signs, the oracle validates traffic rules like stopping at a stop sign or driving at a priority sign. This also applies if the traffic lights are off because then, the traffic signs on the poles control the traffic. But also general specifications like test case timeout, damages on cars or driving off the road must be validated all the time. The difficulty here was to define a set of rules when a test case should be considered as failed or succeeded.

2 Background

My thesis aims to stress fuel-inefficient driving behaviour of autonomous cars. Self-driving vehicles that consume more fuel than necessary won't be accepted by society since they're supposed to drive better than humans in all aspects. To recreate real-life scenarios which tend to maximize fuel consumption, I'm using a procedural content generator which creates scenarios that feature elements commonly found in urban settings, such as traffic lights and traffic signs at controlled intersections. To provide context for my work, I describe how cars convert fuel for driving and how traffic lights work. I also list the necessary traffic regulations that were used in this thesis. Finally, I explain the functionality of simulators and introduce procedural content generation.

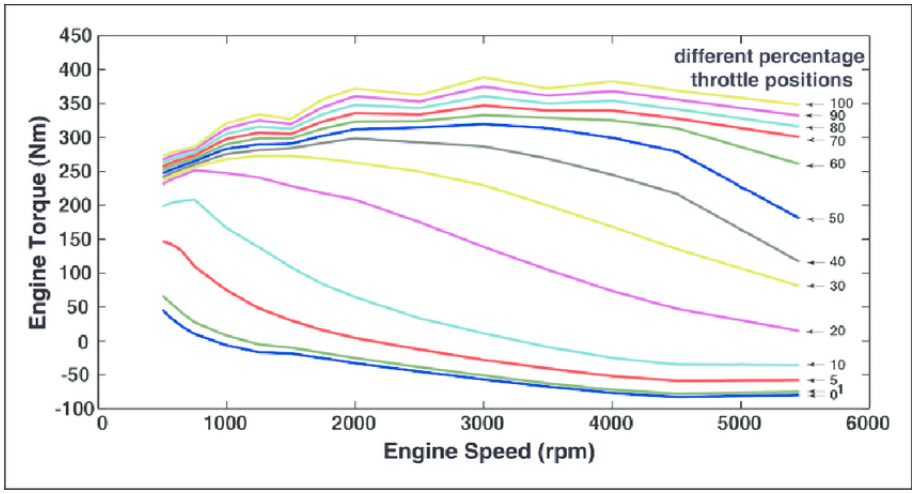

Figure 1: An example engine torque map. The inputs are the RPM and throttle and the output is the engine torque. For example, the highest torque is by pushing the throttle 100% through at 3000 RPM [44].

2.1 Fuel Consumption

Gasoline and diesel vehicles use internal combustion engines. They differ whether the air/fuel mixture is ignited by a spark (gasoline) or by compression (diesel) [56]. In both cases, fuel is injected into the combustion chamber for ignition, causing an explosion which moves the pistons of the engine's cylinders down. This creates the mechanical energy used to move the gears, which then move the wheels [12].
Electronic Control Units (ECU) are electronic devices which get multiple inputs of the car and compares them with stored onboard data. The ECU assesses the inputs and decides on an optimal output for the system. The most important ECU type for this thesis is the Engine Control Module (ECM), which controls the injection and ignition of fuel inside the engine. The ECM takes a lot of inputs from the car, e.g. temperature, engine speed and gas pedal position, and tries to find an optimal way to manage the combustion system like keeping the optimum air to fuel ratio [46]. The throttle controls how much fuel mixture should be injected into the combustion chamber, the harder the throttle is pushed, the more fuel the car consumes [52].
The ECM takes the RPM (rotational speed of the engine) and throttle as inputs and compares them with engine maps, which outputs the torque (a metric for the engine turning force) [21, 14]. Figure 1 shows an example. The torque and RPM can be used to determine the Brake specific fuel consumption (BSFC), a metric to assess the efficiency of internal combustion engines [62]. Figure 2 shows an example RPM-torque-to-BSFC-map. To reach the optimum fuel-efficiency, the RPM and torque must be in certain ranges, for example between 90-140 Nm and 1500-3000 RPM in Figure 2. And as I explained before, the engine torque is dependent on the RPM and throttle. This shows why it is important to shift up the gear as soon as possible because the fuel-efficiency will stay in the optimal area. Additionally, fuel conversion efficiency maps show when to shift the gear for optimal fuel-efficiency. To summarize, fuel consumption is dependent on how hard and at which RPM the driver pushes the throttle.

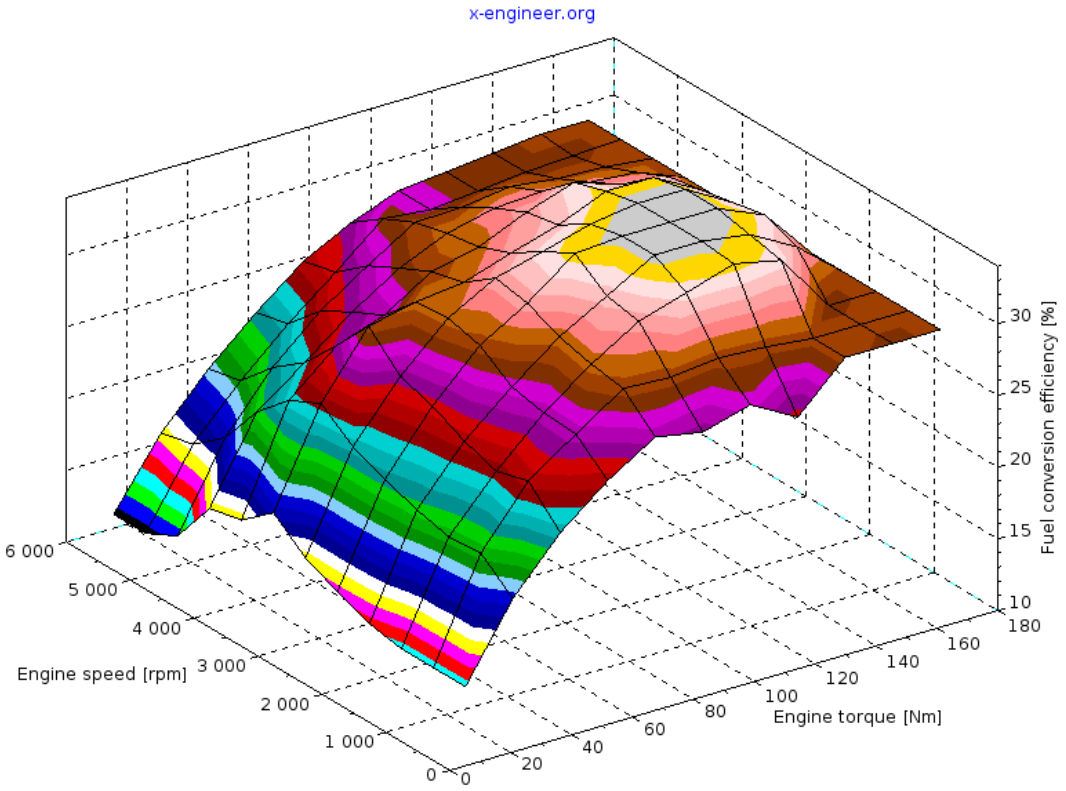

Figure 2: An example fuel conversion efficiency map. The inputs are the RPM and torque and the output is the Brake specific fuel consumption (BSFC), which assess the the efficiency of an internal combustion engine. For example, the highest efficiency is between 90-140 Nm and 1500-3000 RPM [62].

In this thesis, I neither created an engine torque map, nor a fuel conversion efficiency map due to time constraints. Instead, I followed expert suggestions [42] to determine general RPM ranges to shift the gear. I distinguish between diesel and petrol since the engines run at different RPM ranges. Diesel-powered engines need to shift at lower RPM ranges than petrol ones. For diesel engines, this lies at 2000 RPM and for petrol ones at 2500 RPM. Independently of having a fuel map or not, a car driving with too high or low RPM values is very inefficient and consumes too much fuel.
There are many aspects which affect fuel efficiency. On one side, there could be faults on the car itself, e.g. low inflated tires, flaws in the engine or using the wrong motor oil. Another side is environmental properties like driving uphills, bad terrain or traffic jams. These can reduce fuel efficiency as well because they force the car to drive in suboptimal environments which increase the consumption. In my thesis, I focus on bad driving behaviour because this allows assessing fuel-efficiency of self-driving cars. Under this category falls driving with a low gear and high RPM, pushing the gas pedal too hard and brake shortly after again or not turning the engine off while standing on intersections. In this work, I assumed that the ego-car has no malfunctioning components and the environment is in an optimal state, allowing to focus on assessing the driving behaviour only. Optimally driving on a road can be modelled as a sequence of control actions that the driver implements. If a driver does something unnecessary, e.g., accelerating

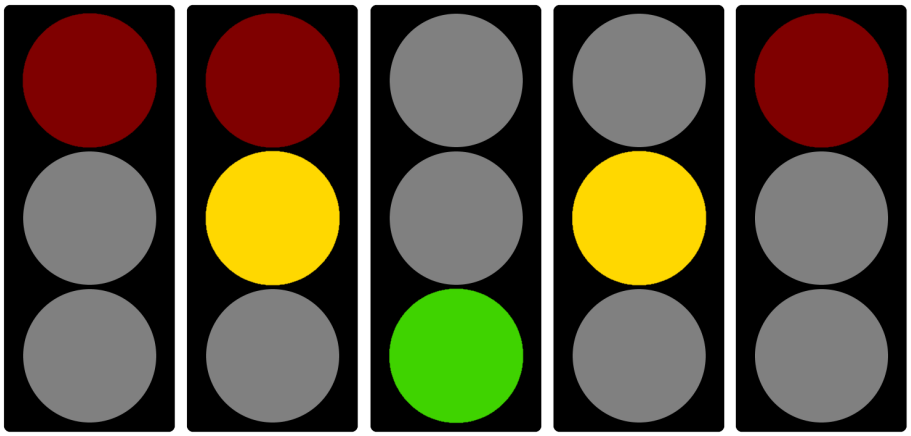

Figure 3: A complete traffic light cycle according to the German "Richtlinien für Lichtsignalanlagen" (traffic light guidelines) [24].

aggressively after the traffic light turns green, the resulting driving style is inefficient and might result in consuming more fuel than necessary and possibly over-using the car's components. In this example, the correct behaviour would be accelerating slowly because the relation of the consumed fuel within this period is way bigger than the saved time that the driver could achieve with this action [17]. Although there are endless possibilities to assess fuel-efficiency, I focused only on a handful of metrics.

2.2 Traffic Lights

Traffic lights control the traffic flow at intersections. There is always at least one lane that is allowed to drive and at least one lane which is not. A green light means that the driver can start driving or keep driving. In contrast, a red light indicates to stop at the intersection. The yellow-red state means the driver should shift to the first gear and prepare to drive on but he/she should still wait until the light turns green. The yellow state indicates that the traffic light is about to turn red. The driver must stop if there is enough distance to safely stop; if he/she entered the intersection or distance is too small then it wouldn't be an infraction to drive through the junction. In this thesis, I refer to the traffic light cycle in Figure 3, the same one is used in Germany [24]. Since this is the first step towards a more complete solution, I consider only the basic traffic lights and no reverse lanes, left/right arrows, etc. However, they are part of my ongoing work.
In addition to the regular cycle of traffic lights, I consider two exceptional, yet completely possible situations. The first one is the flashing yellow light. In Germany, traffic lights at less travelled intersections switch in the evening or night to the flashing yellow light mode. This is beneficial because there are fewer traffic participants on the road at this time and hence, improves the traffic flow because drivers don't have to wait until the traffic light turns green. The second one is off-traffic light. Traffic lights can be turned off for the same reason as the flashing yellow light, to save electricity but they also could be broken. I didn't consider other configurations like broken lamps or differently coloured lights, despite they might be possible.

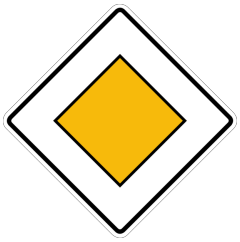

Figure 4: Priority Sign [3]

Figure 5: Yield Sign [54]

Figure 6: Stop Sign [15]

2.3 Traffic Rules

In this thesis, I considered some of the official traffic regulations [43, 64, 10] as specifications to implement test oracles. Since I'm using some traffic signs as well, I explain their rules in this chapter.
A priority sign (Figure 4) [43, 64] gives a driver the right to drive at the intersection first and drivers on the opposite street must wait until it's safe to head out.
In contrast, the yield sign (Figure 5) [43, 10, 28] tells a driver to give right of way at the following intersection until the road is clear. If a driver facing a yield sign takes the right of way of another one facing a priority sign, then this usually results in accidents, fines and penalty points. According to German road traffic regulations, drivers are only allowed in exceptional cases to give up their right of way.
The third sign in the set is the stop sign (Figure 6) [43, 10, 28]. The official rule is to stop in front of the intersection completely, wait for at least three seconds and start accelerating when the road is clear.
The speed limit within a town in Germany is 50km/h. I assume that all traffic participants in my test generator stick to this limit.

2.4 Procedural Content Generation

In this thesis, I generate virtual roads inside a photo-realistic and physically-accurate driving simulator (see Section 4.1.1). Also, a simulator allows changing various properties in many different ways, e.g. routes of traffic participants or geometry of the road. This reduces costs and time to create new test scenarios compared to real life.
Game engines nowadays can simulate driving physics precisely and provide photo-realistic graphics as well. They found use cases in computer vision research [60, 11, 8] while developers sometimes provide SDK tools (software developing kit) to facilitate operating the game engine. These engines are frameworks to build video games and consist of many functional parts like the rendering engine which generates images from two-dimensional or three-dimensional models or the physics engine for approximating physical systems. Each engine needs an accurate description of what they should simulate, e.g. the speed and mass of a vehicle or the polygon mesh to represent an object. Game engines can decide how accurate they want to simulate physics because this depends on the computation power and affects the gameplay. Developers can then create levels and rules with the game engine under the hood.
The gaming industry utilizes a technique to algorithmically create game content with limited or indirect user input called procedural content generation [61]. This content can be anything:

maps, levels, textures, etc. With this method, one can generate endless possibilities of game content. In this thesis, I use procedural content generation to create urban-like scenarios. Each iteration generates a completely new scenario. But using this technique comes with many challenges. One aspect is that generated scenarios must be valid, and this becomes harder with more features. For this purpose, a set of rules which define a scenario as valid must be defined, e.g. roads and road segments should line up, other traffic participants must stay on the road, traffic lights and signs should be placed outside the road, and so on. Another challenge is to choose the right configuration and algorithms to create such content. There are many possible ways to generate road networks or to simulate traffic.

3 Methodology

In this chapter, I present the ideas and the algorithms that implement the approach I propose for testing the fuel-efficiency of self-driving car software.
The test generator part (Section 3.1) introduces the definition, generation and validation of test cases and explains all features in-depth. To create a simulation, I need to build the environment. I start with roads and geometries, which are connected to create a whole road network. I decided to create only some specific scenarios. These include roads with varying curvature and a small set of intersection designs. Roads create connections between intersections and can be used to validate most of my fuel-efficiency scoring oracles, e.g. accelerate-and-stop behaviour. Intersections challenge the ego-car to make the right decisions, for example, it must stop at red lights or keep driving at priority signs. Intersections also allow to validate scoring oracles but in a different way like brake intensity while approaching the junction or the general behaviour while turning. In my opinion, these two scenarios are enough to validate fuel-inefficiency of self-driving cars and to represent urban-like roads. The other features like traffic participants, traffic lights and signs extend the urban-like environment and additionally contribute to the scoring or test oracle, e.g. traffic lights to stop the car at intersections and traffic for giving the ego-car a reason to stop at yield signs.
In the test oracle chapter (Section 3.3), I explain my oracles which define a test case as failed or succeeded. They consist of traffic rules, general properties like simulation timeout, damage and success points. These oracles are enough to classify whether an ego-car can successfully finish an urban-like scenario or not.
Section 3.2 elaborates on my scoring oracles which assess the fuel-inefficiency of self-driving cars. For this purpose, I use the following five oracles:

- RPM Infraction
- Throttle Infraction
- Brake Infraction
- Engine Idle Infraction
- Accelerate-and-Stop Infraction

They are taken from expert suggestions [51, 18, 42] as well as common sense.

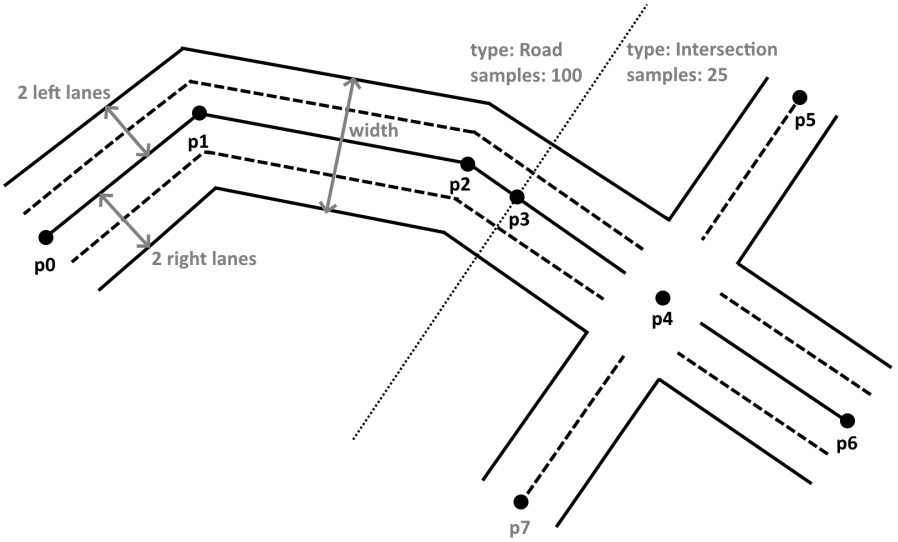

Figure 7: Demonstration of road properties.

3.1 Test Generator

3.1.1 Road Definition

Simulations require to define some abstraction of roads. I decided to use 2D coordinates because they facilitate the computation heavily. Despite that, it is still relevant to test fuel-efficiency with flat roads. The direction of the road is implied by the sequence of points in the list. My roads have the following properties which I exemplify in Figure 7:

- Point list: A list of two-dimensional points
- Number of left and right lanes
- Width: Width of the whole road
- Samples: Number of points used to interpolate the road
- Road type: Can be "normal" or "intersection"

I chose a minimum of one lane for each direction and a maximum of two. I'm not focusing on one-way streets, hence the minimum of one. I decided on a maximum of two because this allows over-taking and lane switching. I focus only on the second point but since my test generator creates such tests, both behaviours can be tested.
The width parameter defines the width between the left and right side of the road at every point. It is possible to assign each point its width but this would make some tasks much more complex, e.g. drawing the side road markings or defining waypoints for traffic. This could be also done in future work. The width is dependent on the number of left and right lanes. One lane can take the width of four of five. Through trial-and-error, I determined the optimal road width and

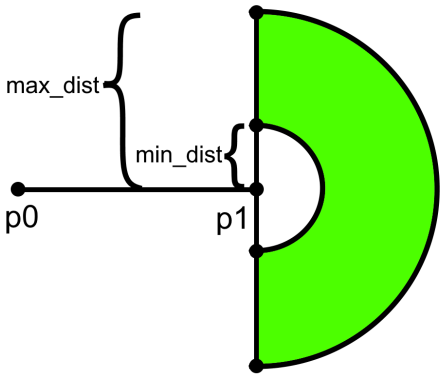

Figure 8: The green area contains valid points.

these were the best ones. The width per lane is chosen randomly for the whole road and then multiplied with the total number of lanes.
Each road holds its number of samples for interpolation, allowing users to decide how fine-grained the roads can be interpolated. I explain this in Sections 3.1.3 and 3.1.6.
The road type facilitates several functions or makes some even possible. This is relevant in the following sections.

3.1.2 Road Generation

A road network consists of multiple roads. I differentiate here between intersections and normal roads. This section is about the generation of the later one. My test generator creates a new road after each intersection. Otherwise, some functions like placing parked cars wouldn't be possible. I assign the first three points of the first road manually to create a short, straight segment at the beginning of each test case. This ensures that the car spawns fully on street terrain and that the onboard camera sees the course of the road; if the test case starts with sharp turns then the AI might not see the road in front of it. All other roads start exactly where the intersection ends which I will explain in a moment. For each new road, the number of lanes and the width per lane are chosen randomly.

Points are randomly generated and then appended to the point list. To avoid extremely sharp turns I limited the angle range. The sharpest turn can have 90 degrees for both directions. Because it would be unrealistic having very wide streets with 90 degree turns, I reduce the range by 25 for each additional lane after two for both directions. Additionally, I avoided too short and too long distances between two consecutive points. The minimum and maximum distance can be assigned during the initialization process of the test generator. Figure 8 exemplifies this.

3.1.3 Interpolation

Interpolation creates a mathematical function to estimate new data points within the range of discrete known points [69]. In this thesis, this is useful because out of relatively few input points a street with smooth round turns can be generated. An example can be seen in Figure 9. The

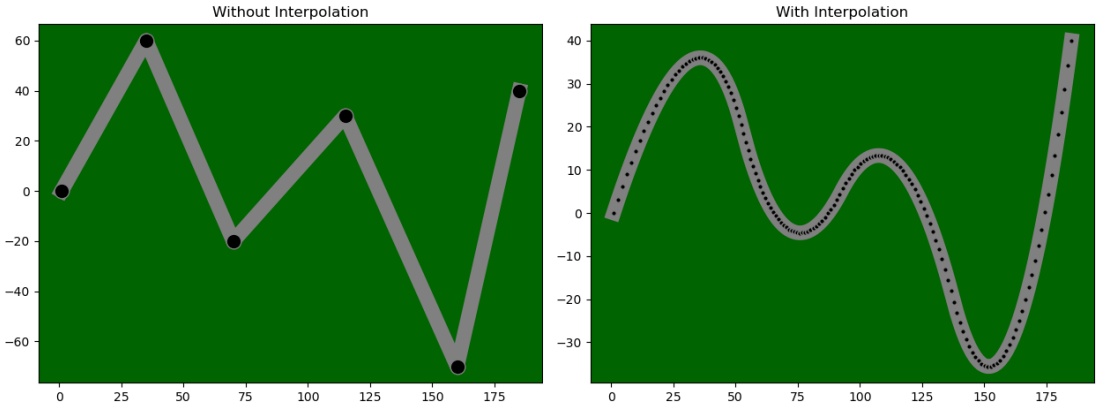

Figure 9: Demonstration of Interpolation.

function generated out of six points 150 new ones.
There are many interpolation types and functions. I found that a spline interpolation would be the most suitable one; I prefer spline over polynomial because it reduces the interpolation error due to its piecewise nature (meaning that it consists of many subfunctions). I'm using the b-spline interpolation [9]. A comprehensive algorithm can be found here [68]. This interpolation function takes a set of points as an input and generates a curve of degree n which starts at the first point of the input and ends with the last point. The lower the degree, the closer the curve follows the polyline of the input dataset. Depending on the use case, there is one drawback: the curve doesn't go through the other points of the set. However, by choosing the correct parameters, the curve can reduce the distance to the points very well. Figure 10 shows a comparison between two spline degrees and how the number of input points affects the interpolation. I decided to take degree two as a default value because the results are satisfying in my opinion.
I'm using the b-spline implementation of the SciPy package, a free and open-source Python library for mathematics, science, and engineering [67]. In my implementation, I take a list of control points as an input and return a list of splines. The number of samples is optional.

3.1.4 Intersection Definition

I define two types of intersections: three-way and four-way intersections. As the name suggests, there are three or four connected roads. There are a lot more possibilities, but these two were the most simple ones. Each direction is represented by one road, so a four-way intersection consists of four roads. There is always a point in the centre connecting all other roads and also one point leading to the direction. In a four-way intersection, there are a total of five points. Hence, each road is straight. This is a very simple intersection design, yet enough to create a scenario where the ego-car must handle the case of stopping and turning while facing traffic participants as well as traffic lights and signs.
Three-way intersections must also define one of the three layouts. Figure 11 shows examples of possible intersection combinations.
The test generator decides on one direction and starts appending new segments to the corresponding road.

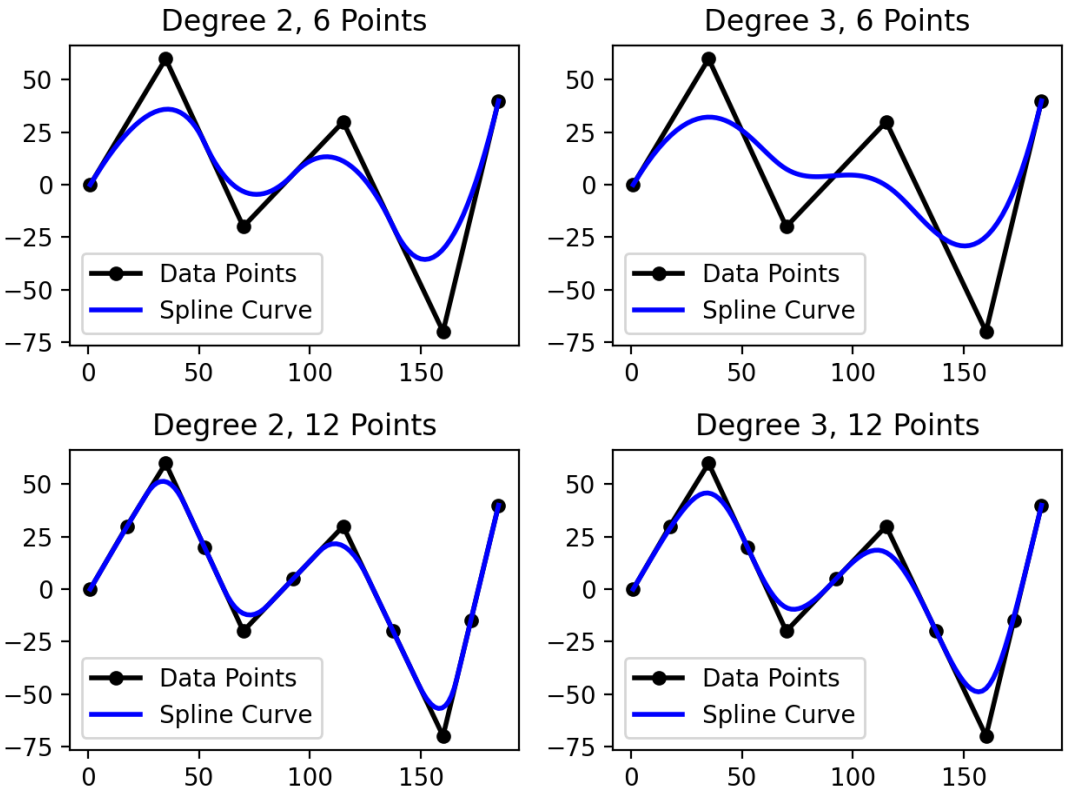

Figure 10: Comparison between degrees and the number of input points.

3.1.5 Intersection Generation

After the layout and direction of the intersection are known, the corresponding coordinates must be generated. For this purpose, I want to clarify the following: the direction from where the ego-car approaches the intersection is the lower one; from there, it can turn left, right or heading straight (upper direction). The road on the lower direction stays for all intersection types the same. The algorithm is visualized in Figure 12.
The first step is to get the centre point. This can be achieved by creating a line between the penultimate and the last added points and resizing it. The distance to the middle point is assigned during initialization and stays for all intersections the same. After that, the other points can be generated by rotating and resizing the line between the middle point and the lower direction point, with the first one as the rotation origin. Each direction has a rotation range to generate a variety of intersections. For straight segments, I chose a range between [-10,10] degrees, for the left and right one [70, 110] and [-110,-70] degrees. The road on the upper direction gets the same number of lanes and width as the one on the lower direction. The roads on the left and right get randomly selected values. However, the number of left and right lanes must be switched, else the roads will become point symmetrical. After that, the algorithm generates roads out of the created points and appends them to the road network. They all get the type-tag intersection. Now, new points can be generated and appended to the intersection road which

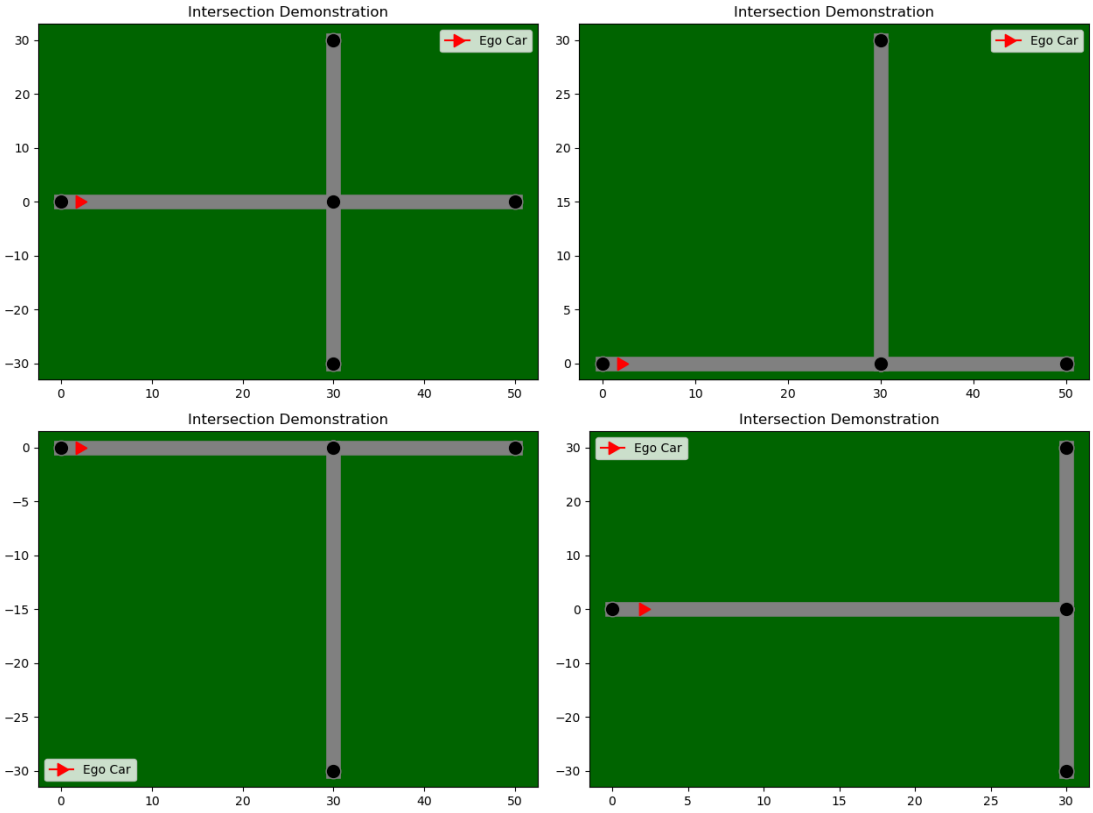

Figure 11: A demonstration of possible four-way and three-way intersections.

leads the direction.
The design of junctions is pretty limited at the moment. Future work will generate a richer set of intersections.

3.1.6 Validity Checks

Before new points or intersections can be appended to the road network, they must go through a set of checks to ensure that the test case is valid. I distinguish between road pieces within intersections and outside of them, each one has its definition of invalid roads.

Roads Outside of Intersections
After generating a new point, I create lists of lines by connecting two consecutive points of a road's point list. I repeat this for the whole road network. Then, I create another line between the last added point and the generated one. A test case remains valid when the generated point doesn't lead to any form of intersecting roads, so this line must not intersect with any other polyline. Otherwise, the point will be discarded. Figure 13 shows an example of the whole procedure.
To avoid partial-intersecting roads, the width in every point must be checked as well. I interpolate the road network for this purpose to reduce the gaps between points, the more points

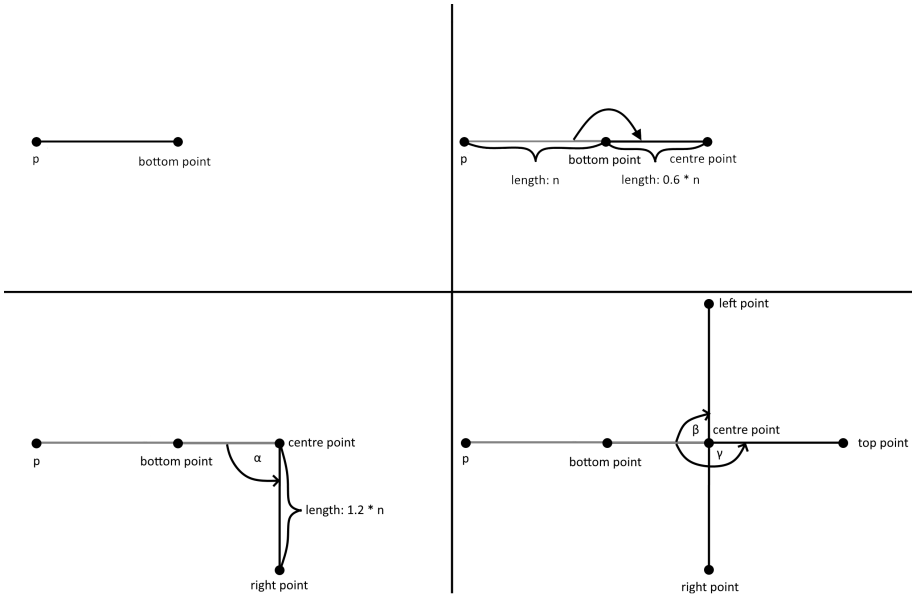

Figure 12: Algorithm for generating an intersection.

the more reliable the validity check. With the splines, I generate lists of polylines again as well as width lines. These lines represent the width of the road, starting from the left to the right roadside. To get them, I rotate the polylines in every spline (rotation origin) by 90 and -90 degrees and resize them to the length of the road width. If a width line intersects with a polyline that is not its origin, then this would result in an invalid test case because, in the simulation, there would be two roads segments intersecting partially. Because a new point changes the whole interpolation curve of the current road, each width and polyline of the current road must be validated, resulting in $O(n^4)$ steps. This is the reason why I assign roads 100 interpolation samples and intersections only 25, this increases the performance a lot and the value is high enough to detect invalid intersections. Again, Figure 14 shows the algorithm visually.
If neither any polyline intersects with another, nor any width line with a polyline, then I append the new point to the road network.

Roads Within Intersections
Now it's about checking whether adding an intersection to a valid road keeps the road network valid or not. Because junctions consist of two opposite roads, I need to check for intersections between the two new polylines (which represent the junction) and any other polyline. This is the same algorithm as in Figure 13. I apply this algorithm consecutively because otherwise, my function detects an intersection between the two new polylines and the test case gets marked as invalid.
Again, to avoid partial-intersections, I apply the second algorithm (Figure 14) twice to check for intersections between poly- and width lines.
If the validity checks are positive, the intersection gets appended to the road network.
Figure 15 shows how an example of a randomly generated, valid road network. Each colour represents a new road.

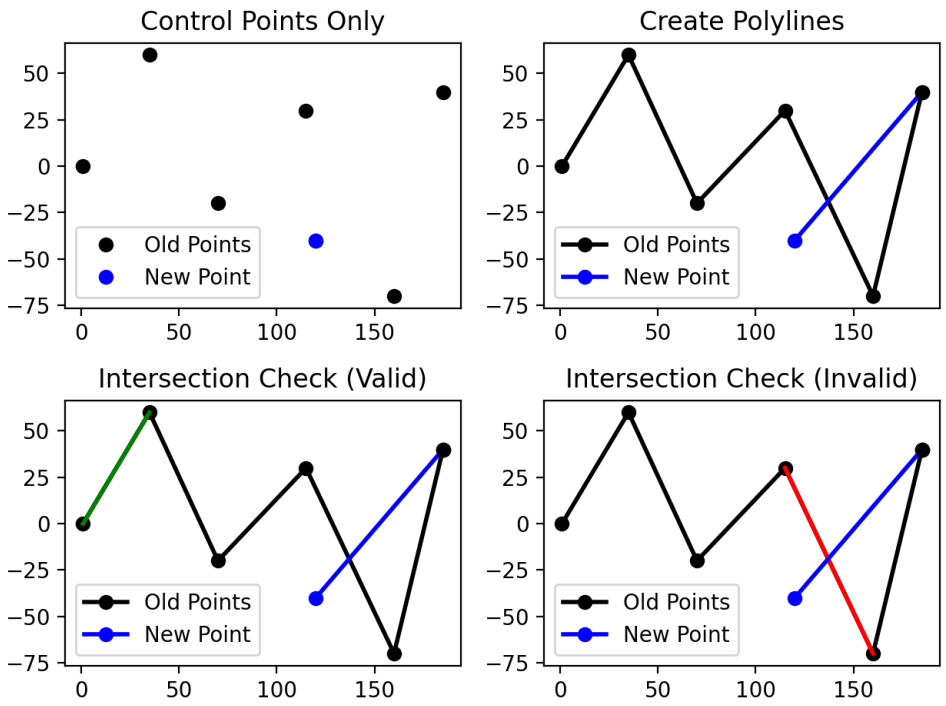

Figure 13: Algorithm for checking polyline intersections.

3.1.7 Traffic Lights and Signs

After an intersection was successfully added to the road network, the test generator adds traffic lights and signs to each existing direction. Traffic lights and signs must not stand on the street and as close to the intersection as possible. At this stage, the width of both roads, as well as the points defining them, are known, so I can compute the position of the objects accurately. Figure 16 visualizes an example to determine the position of one traffic light/sign. The first step is to create a line between an endpoint of the intersection and the middle point (from B to A). After that, I scale the line to the width of the opposite road and add a small offset to ensure the position will be outside the street (scale AB to AC with resize factor x). Then I rotate the line by 90 degrees clockwise on C and resize them again with the width of the current road (with the resize factor y). The same algorithm applies the lines RA, SA and TA as well. Because the angle isn't always perfectly 90 degrees between two roads, the object might stand on the street. To avoid this, I added a small offset to resize factor x which depends on the angle. The values were experimentally determined. I tried several methods, e.g. creating two vectors and rotating by the angle between the lines BA and RA but this approach returned the best results. Another important point is to ensure the correct rotation of the object, otherwise, traffic participants won't see it. This can be done, for example, by calculating the angle between BA and a horizontal line starting at the centre point.
After the positions are determined, the test generator decides whether to place a traffic sign or traffic light. This depends on the number of lanes. If the current or opposite road contains more than one lane per direction, then this intersection gets traffic lights. The reason for this

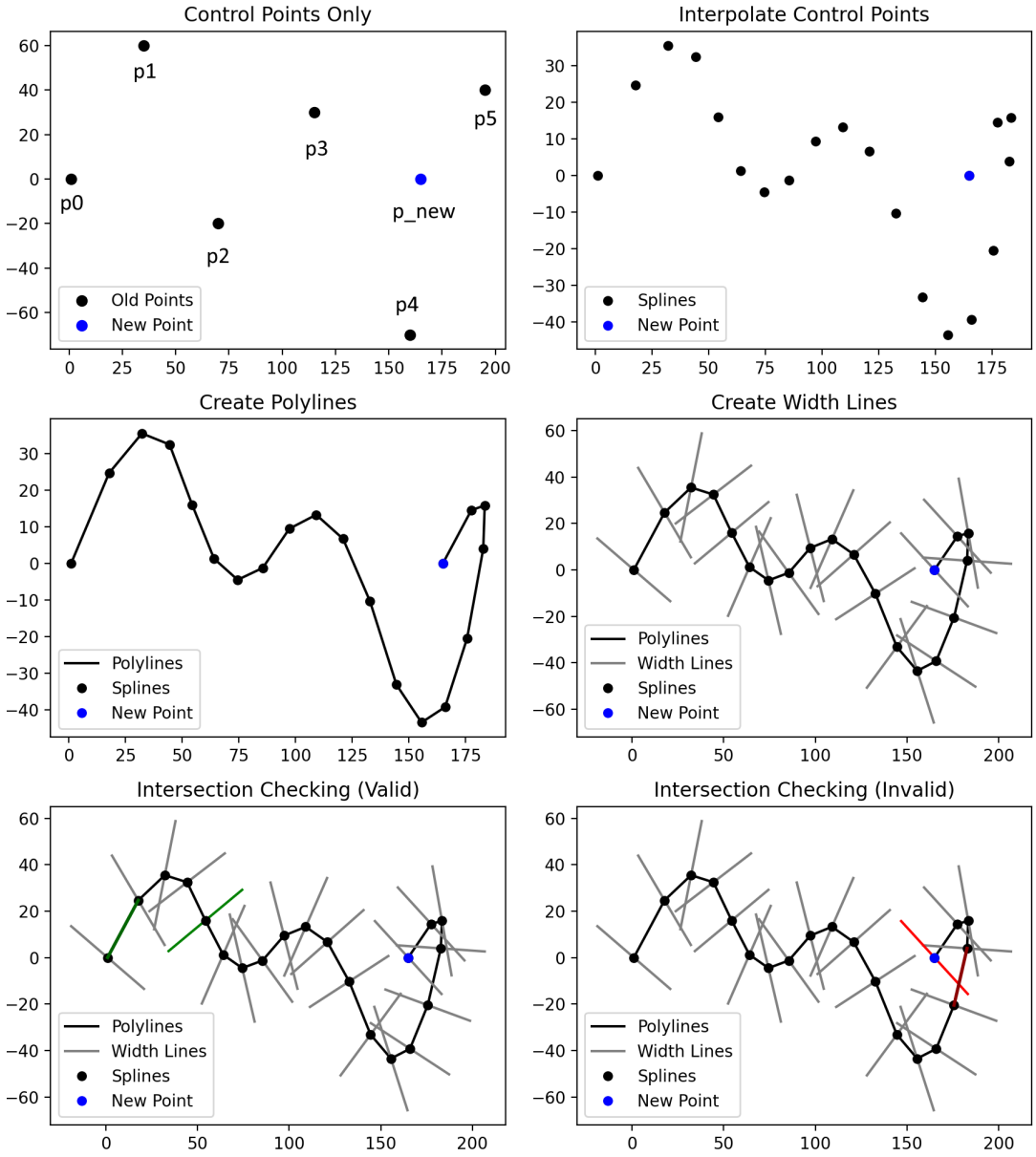

Figure 14: Algorithm for checking polyline and width line intersections.

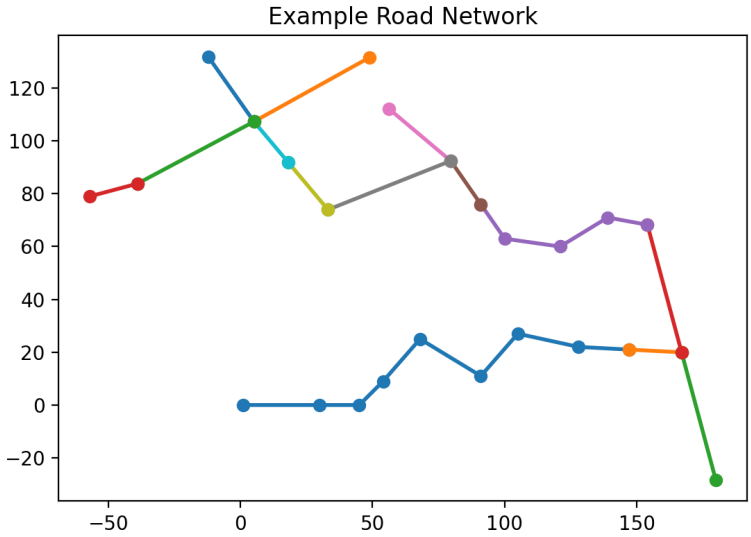

Figure 15: An example of a valid road network with intersections. Each colour represents one road.

is because intersections with multi-lane roads usually have a higher traffic density, which is preferably controlled by traffic lights. Intersections with traffic signs and multi-lane roads exist, but they are very unlikely, especially in urban environments. I couldn't find any guidelines for placing traffic lights and signs on intersections, so this explanation is based on my experience. If the right side of the road contains one lane then it gets a single traffic light, else a double which contains two lights and is way bigger. This is common in urban environments because otherwise, drivers standing at the left lane of an intersection might not be able to see the traffic light. If both roads contain one lane per direction, then the possibility of having traffic lights and signs is equal. The probability of facing a stop or priority sign is also equal. To keep road networks simple, the bottom and top direction share the same sign as well as the left and right direction.
Traffic lights get a yield or priority sign attached to the pole because when they're not working, the signs control the traffic. The signs are again equally possible. Also, one out of three modes is randomly assigned to the traffic lights of one intersection (Section 2.2). I will describe the manual mode in detail. At the moment, I implemented only two sequences; in the first one, the initial colour of the traffic light stays red until the ego-car enters a certain point. This one is called a trigger point because if entered, it activates a certain event or code. This is very common in video games and allows to create reacting environments. Now, if the ego-car enters this trigger point, the traffic light switches to yellow-red and after 1.5 seconds to green. The state stays like this until the test case ends. The German road traffic regulations [10] states that the switch from yellow-red to green must last at least one second. I increased this value to 1.5 because intersections with a high traffic density can take longer to switch to green, so my decided value falls in between both sides. The exact behaviour is not defined by the test generator but by the algorithm converting the test cases to simulation files (Section 4.4.2).
In the second sequence, the initial colour of the traffic light is green and after entering the trigger point, it switches to yellow and then to red. After seven seconds, which is the time the first type of traffic participants (Section 3.1.9) needs to drive from its spawn to endpoint, the light turns to yellow-red and after eight to green. This time, the switch from yellow-red to green is

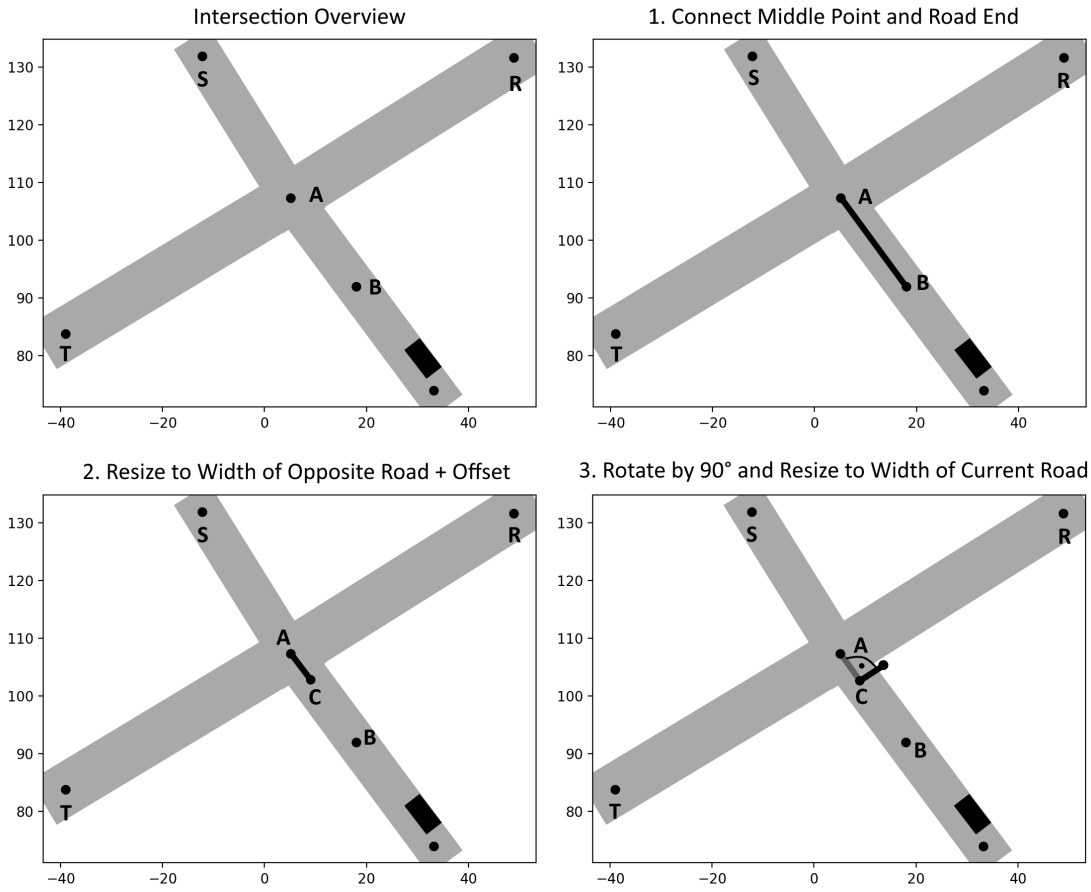

Figure 16: Example to get the position of a traffic light/sign.

shorter to add more variety to my traffic lights implementation. In future work, I will leave these parameters open to the user to have more control over the scenario. There are way more possibilities to control traffic lights, e.g. a green traffic light which does not switch its state to red, which will be also added in my ongoing work. In this green-to-red sequence, the ego-car must stop at the intersection, otherwise, it would drive through red lights and this counts as an infraction (Section 2.3).
The position of the trigger points is always at the last generated point of the list, e.g. like the bottom right point in Figure 16. It is also worth mentioning that only the traffic light which faces the direction of the ego-car has working light switches, the others are turned off because they are not visible in the front camera. Figure 17 shows a test case that this algorithm generates.

3.1.8 Urban Scenario Generation

In this section, I describe the general algorithm to generate an urban-like scenario.
I'm following an incremental approach where I start with three initial points with randomly

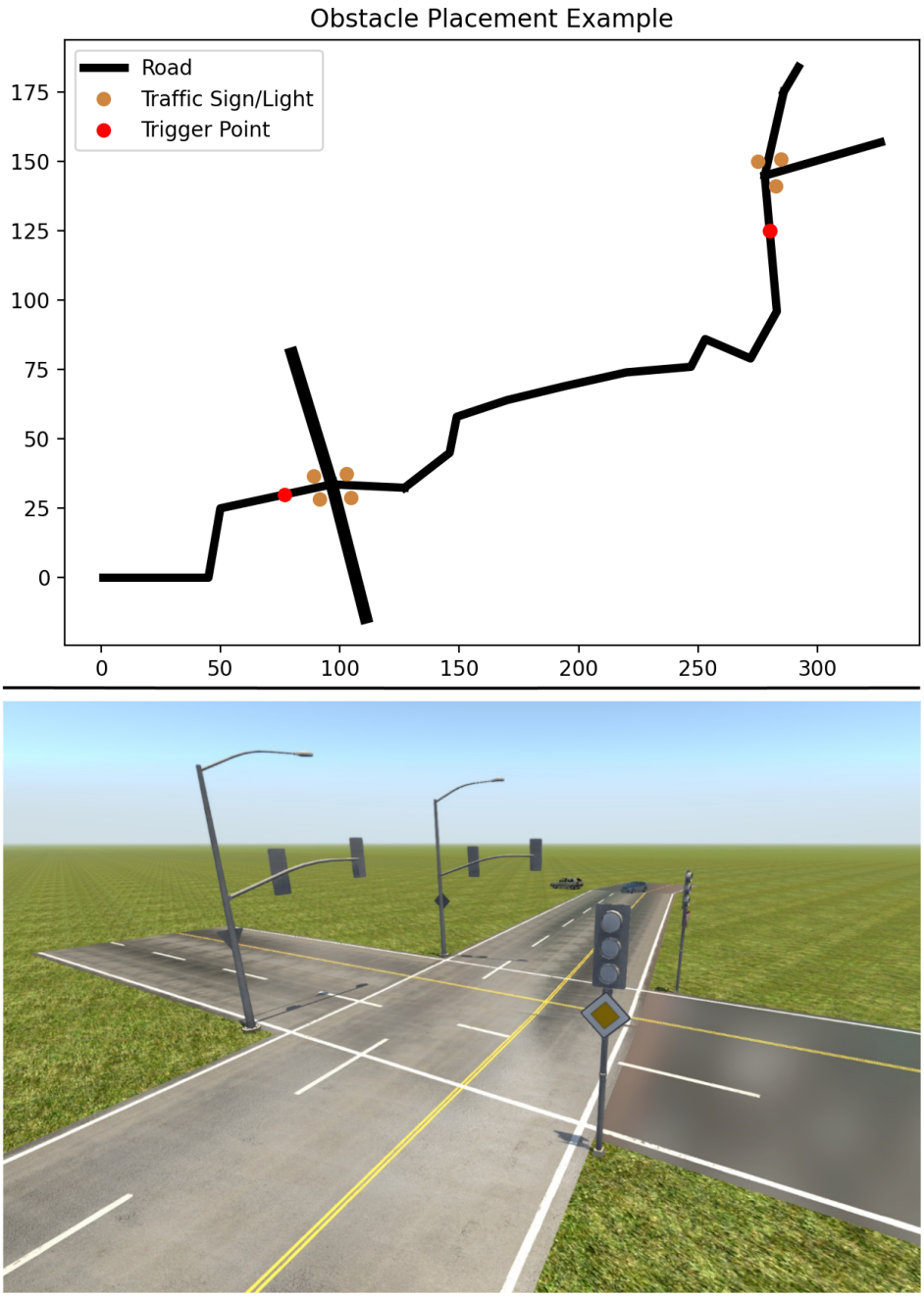

Figure 17: Top image: randomly generated test case with visible traffic lights/signs and trigger points positions. Lower image: randomly generated intersection with traffic lights in the driving simulator BeamNG.research [5].

chosen road properties (Section 3.1.2) and append either new points or an intersection (3.1.5) to the road network. I chose the probability of 25% for a junction, while new points are generated in every iteration. I avoid having two consecutively intersections because this would create monotone test cases. So, there must be a road between two consecutive intersections. Also, every test case must have at least one intersection to stress the ego-car to violate every scoring oracle (Section 3.2). That's why I force one if the number of points plus intersections reached the limit otherwise I consider it as invalid and the algorithm restarts. The algorithm terminates when the number of points plus intersections reach the limit (I chose 16 to avoid long test cases) or the generator couldn't create valid points/intersections after 20 consecutive iterations (value is configurable). I define the minimum and the maximum number of generations which must be fulfilled after the loop breaks.
The valid test case contains the lanes, a success point where a test case is considered as succeeded on entering, traffic lights/signs, trigger points and the time of day. The success point is just the last generated point. The time of day is randomly chosen on a scale between zero and one. After that, I interpolate all roads for the following tasks and merge roads following the same direction to avoid interrupting segments. The mentioned tasks include adding traffic participants (Section 3.1.9) and parked cars (Section 3.1.10).

3.1.9 Traffic Participants

Besides the ego-car, I implemented two types of traffic participants called non-ego-car 1 and non-ego-car 2. The first one appears only at intersections and drives to a randomly chosen point which does not follow the ego-car's direction. The second one drives between two intersections on the opposite lane to pass by the ego-car. Both traffic participants have way- and spawn points. Waypoints are common in video games, a moving object has to traverse through a set of points in a specific order. Hence, I can design the test to move the vehicles in a predictable way. Spawn points are areas where objects, players or NPCs (non-playable character) are created or teleported to, e.g. car z should spawn at point (x, y).

Ego-car
During test generation, I stored the directions of travel at intersections, e.g. turn left at the first intersection, keep going straight at the second intersection, etc. This information is necessary for a self-driving car software because otherwise, the ego-car doesn't know which direction it should take at an intersection.
In case that the ego-car is not a vision-based self-driving software but an AI which follows waypoints (for example to test a traffic light detection system live during driving), I provide a list of those points from the first to the last generated point. During test generation, I stored the roads which must be driven by the test subject, I call them "ego-lanes". For each ego-lane, I connect all points with lines to form a polyline. Because this polyline represents the centre of the road, I'm shifting the line with the distance x to the right so the ego-car can follow the correct lane. Distance x is dependent on the road width and by the number of lanes. By changing distance x, I can also create lane switches, for example, to turn left at a two-lane road at an intersection.

Non-ego-car 1
The first of the two non-ego-car types always spawns at intersections on the opposite road or lane with respect to the ego-car's lane. The ego-car must face a stop sign, a red light or a yield sign on the pole of a non-working traffic light, otherwise, non-ego-car 1 will not spawn. Non-ego-car 1 has three possible spawn points and two endpoints on four-way junctions, on three-way ones

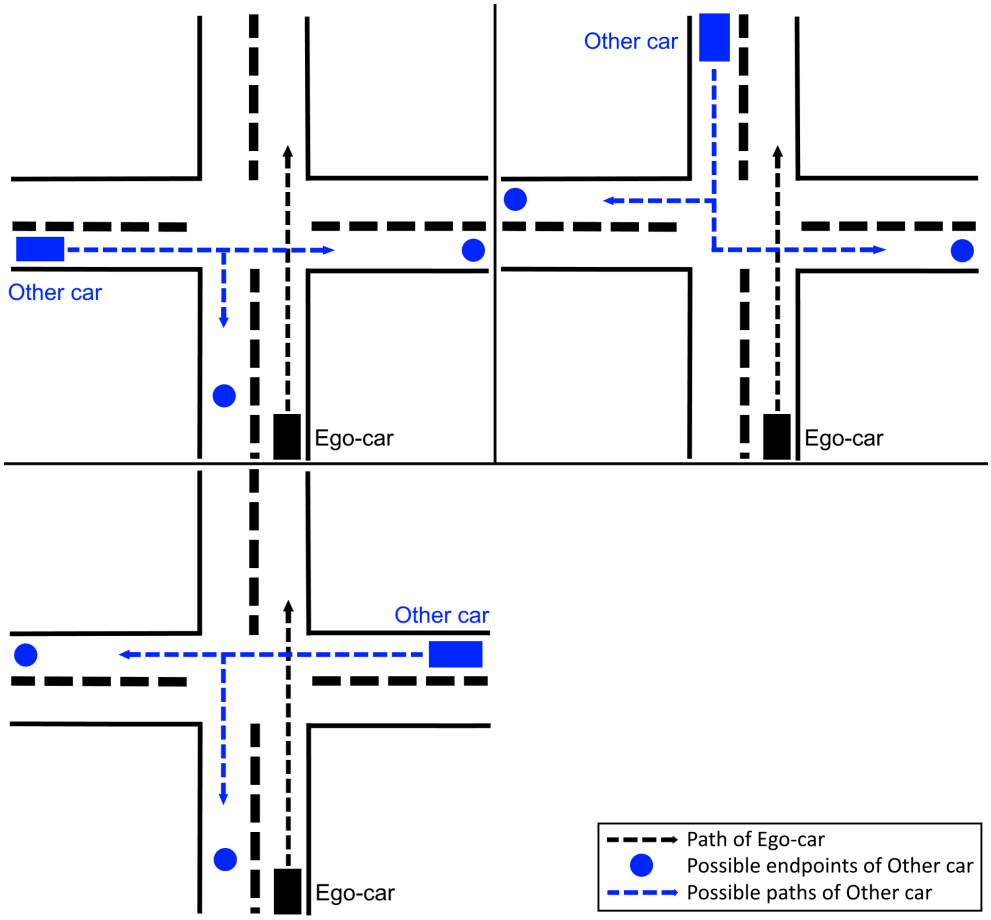

Figure 18: Spawn/end points and paths of the first participant.

there are two and one. Figure 18 exemplifies the algorithm.
The spawn and endpoints are the edge points of the intersection roads. The spawn and endpoint are randomly chosen. The driving path consists of driving from the spawn to the centre point of the intersection and from there to the endpoint. I can also alter the speed to increase the variety of traffic participants.
The traffic participants spawn and start driving when the test subject enters a trigger point. They are handled in the same manner as traffic lights but contain different information. Each trigger point knows which vehicle should be triggered, where the non-ego-car should spawn and its orientation. They're placed when the ego-car is about to approach the intersection, so whenever a "normal" road ends.
The non-ego-car 1 forces the ego-car to slow down at yield signs, otherwise, both cars would crash which results in a failed test case (Section 3.3). This allows assessing the driving behaviour of the ego-car considering its fuel-inefficiency because the test subject has to brake and re-accelerate again.

Non-ego-car 2
The second non-ego-car type always spawns at the end of "normal" type roads and drives the

ego-car's opposite way. So, this vehicle has only one possible spawn and endpoint. Non-ego-car 2 spawns and starts driving when the ego-car enters a trigger point. These trigger points are placed when normal-type roads start, so whenever the ego-car leaves an intersection.
Non-ego-car 2 does not affect the fuel consumption of the ego-car, but in my ongoing work, I will create experiments to see whether a vision-based self-driving AI behaves differently when a car on the opposite lane approaches it. For example, the AI recognizes non-ego-car 2 and slows down, which would then affect the fuel consumption because the ego-car has to accelerate again. This is an assumption which needs to be evaluated.

3.1.10 Parked Cars

My test generator generates parked cars because they are common in urban scenarios. As non-ego-car 2, they don't affect the fuel consumption of the ego-car, but I will evaluate whether vision-based self-driving AIs behave differently on test cases with and without parked cars, which might affect the fuel consumption. Right now, they are just a visual feature.
Parked cars represent non-driving vehicles standing beside the road. According to the German traffic regulation, drivers must keep a distance of at least five metres to the intersection while parking. Also, drivers must always park their cars in the road's direction of travel [2]. My algorithm generates parked cars according to those rules.
I defined the following alignments: parallel to the road, rotated by 45 and 90 degrees. Also, the cars can be parked on both sides, on one or none. In the current prototype, the generation of parked cars is random, as the focus of the thesis is not to find the best placement but to generate various scenarios. The angle determines the offset to the road and the minimum distance between two cars such that no two parked cars collide. Of course, this distance is dependent on the length of the car itself, but in this work, all cars are the same and I know their dimension. To create variety and increase realism, I added noise to the rotation and offset of every single parked car by mutating the reference position and rotation a little bit. Figure 19 shows various combinations of parked cars placed in the test case.
To get their positions, I create polylines between the road points and shift them to the left and right by the half of the road width. Because the roads are interpolated, every interpolation point can be a possible position for a parked car. I consider a parking spot as valid when the parked car is not standing on any road segment and the distance to the previous parking spot is greater or equal to the minimum allowed distance.
After every road is done, the algorithm contains a list of possible parking positions. To create free parking spots, I discard one position with the probability of 40%. Also, I give each car its random colour.

3.1.11 Configurable Parameters

My test generator can be configured in different ways with many optional parameters which were mentioned in previous sections. Here is the full list:

- File name: The file name of all generated XML files
- Traffic: Flag whether to add other traffic participants or not

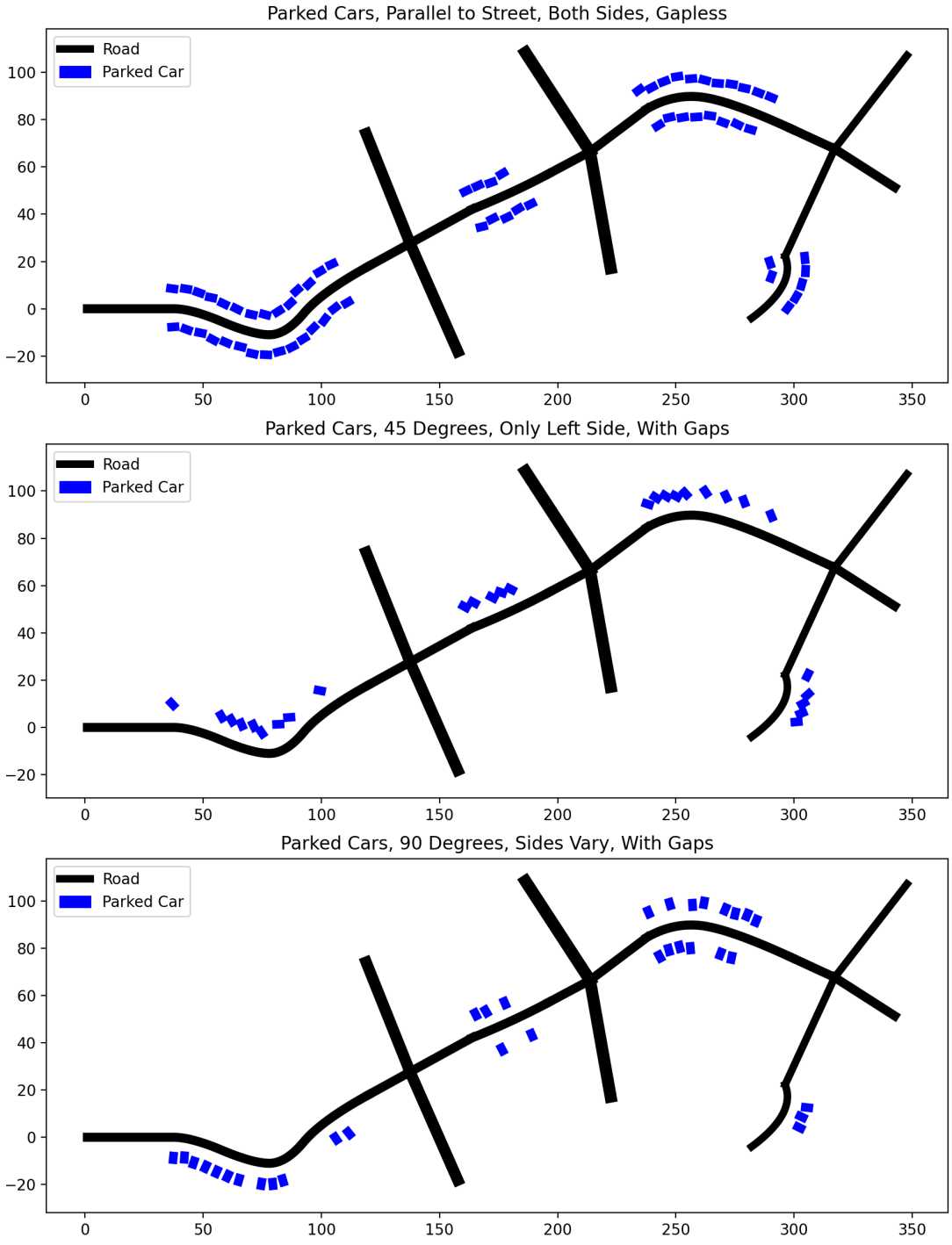

Figure 19: Various combinations of parked cars.

- Spline degree: The degree for the b-spline interpolation curve
- Maximum tries: The maximum amount of consecutive generated invalid intersections or points
- Population size: The number of test cases generated in one run
- Minimum and maximum segment length: I refer to Figure 8
- Minimum and maximum pieces: A range of how many points or intersections (counts as one) a test case should have
- Length of the current road: The length between the points of the bottom and top direction. In the first plot of Figure 11, this would be the distance between the very left and right point
- Length of the opposite road: Distance between the middle point and the left/right point of an intersection
- Length to the intersection: Distance between the bottom point and the middle point
- Maximum number of left/right lanes
- Maximum width per lane
- Intersection probability during generation

3.2 Scoring Oracle

In this section, I present how the scoring function for fuel-efficiency is calculated. Annually, the team of the driving simulation CARLA [16] creates a competition called the CARLA Autonomous Driving Challenge [66]. There, test subjects are scored by metrics which depend on many factors. The score goes up for each completed metre and down for each infraction [47]. Therefore, scoring functions are already in use in the domain of assessing self-driving cars, albeit for different goals. I followed a similar approach but defined scoring oracles that capture the fuel-efficiency of test subjects. For each violation of a fuel-efficiency specification, the scoring function increases, table 3.2 shows the penalties. The score never decreases because my function should only assess fuel-inefficient driving behaviour. Also, this would require oracles which reward fuel-efficient behaviour. The scoring oracle validates the specifications once every second. The simulation creates a snapshot of sensor values and simulation states and passes them to the oracle, which checks them during runtime.

3.2.1 RPM Infraction

For this oracle, it is important to specify the engine type (diesel or petrol) before the test case starts because for diesel engines, the suggested shifting spot lies at 2000 RPM and for petrol ones at 2500 RPM [42]. To define an upper limit, I increased these values by 200 to define a tolerance range, which is big enough in my opinion. It is also important to specify the number of gears because it doesn't make sense to validate for RPM infractions at the highest gear since the

Infraction	Penalty
RPM	1-9
Throttle	1-9
Brake	1-5
Engine Idle while Standing	4
Accelerate-and-Stop	1-8

Table 1: Infraction Table

RPM Range	Penalty
2201 - 2400	1
2401 - 2600	2
2601 - 2800	3
2801 - 3000	4
3001 - 3200	5
3201 - 3400	6
3401 - 3600	7
3601 - 3800	8
$\geq$ 3801	9

Table 2: Penalty points distribution for RPM infractions of diesel engines.

RPM Range	Penalty
2701 - 2900	1
2901 - 3100	2
3101 - 3300	3
3301 - 3500	4
3501 - 3700	5
3701 - 3900	6
3901 - 4100	7
4101 - 4300	8
$\geq$ 4301	9

Table 3: Penalty points distribution for RPM infractions of petrol engines.

ego-car can't shift higher. Therefore, this scoring oracle only validates the RPM, if the current gear is lower than the highest possible gear. So besides the current RPM of the ego-car, the simulator must also provide its current gear as an input.
The kilometres per litre decrease with a higher RPM. But since it would be unfair to penalize RPM values near the red line the same way as to lower values, I increase the infraction points by one for each additional 200 RPM after the threshold, with nine penalty points as a maximum. The height of the penalty can be seen in tables 2 and 3.

3.2.2 Throttle Infraction

Fuel-consumption is highly dependent on how hard a driver pushes the throttle. Manufacturers like Nissan implemented a resistance system so people avoid pushing the gas pedal too hard [51]. According to several other experts, they suggest applying light throttle. I defined 60% push through as the limit. I couldn't find any proven numbers so I experimented with this value.
For this oracle, the simulator passes the current throttle position of the ego-car. The position then gets validated, whether it exceeds the 60% limit or not. The height of the penalty is dependent on how hard the gas pedal is pushed through, with a maximum of nine points. Table 3.2.2 shows the distribution of penalty points. I chose the range rather small so the penalty will quickly rise to the maximum.

3.2.3 Brake Infraction

Brake infractions happen when the test subject brakes too hard right, for example, before turning at an intersection. This also affects the fuel-economy because the vehicle needs to accelerate

Throttle Position	Penalty
$60\% < X \leq 63\%$	1
$63\% < X \leq 66\%$	2
$66\% < X \leq 69\%$	3
$69\% < X \leq 72\%$	4
$72\% < X \leq 75\%$	5
$75\% < X \leq 78\%$	6
$78\% < X \leq 81\%$	7
$81\% < X \leq 84\%$	8
$> 84\%$	9

Table 4: Penalty points distribution for throttle infractions.

again until it reaches the target velocity or the car accelerated before while it should have used the engine brake. Since my thesis doesn't consider emergency braking, I'm not distinguishing between these two properties.
I defined a brake position of 45% as the upper limit. This is highly debatable because on one side braking is highly dependent on the current situation. I argue that in regular cases like my test cases, a smooth driving style is possible without misusing the brake pedal and hence, it's dependent only of the driver. On the other side, it might be necessary to brake hard to ensure the passenger's safety but again, I assess fuel-efficiency and no other properties. The simulator passes the brake position to this oracle, which gets validated, whether it exceeds the limit of 45% or not. The penalty score is equally distributed between 46% to 100%, with a maximum of five points (table 3.2.3).

Brake Position	Penalty
$45\% < X \leq 56\%$	1
$56\% < X \leq 67\%$	2
$67\% < X \leq 78\%$	3
$78\% < X \leq 89\%$	4
$89\% < X \leq 100\%$	5

Table 5: Penalty points distribution for brake infractions.

3.2.4 Engine Idle Infraction

Standing on an intersection with the engine running is counterproductive since the car is not moving but still consuming fuel. Penalizing this behaviour can improve the fuel economy. The right way would be to turn off the engine. This specification doesn't affect the fuel-economy too much and hence, the same counts for the infraction points.
I validate this specification every time the car comes to complete halt. Experts debates at how many seconds of waiting it's profitable to turn off the engine. Some experts say that it's not worth it at all, others name ranges between 10 to 90 seconds. I chose one second because drivers never know when the traffic light turns green. Also, I wanted to demonstrate that my scoring

oracle can spot this infraction because the ego-car stands for about four seconds at intersections in my test cases. To summarize: If the test subject stands for over one second with its engine on, this counts as an infraction.
The simulator must pass the current velocity of the vehicle, a flag which tells whether the engine is running or not and the simulation time. This oracle validates only when the velocity is zero. Then, the oracle stores the simulation time, at which the ego-car stopped completely. If one second passes and the engine is still running, then this counts as an infraction and the scoring function increases by four.

3.2.5 Accelerate-and-Stop Infraction

This is the main reason why cars consume a lot of fuel in urban environments and traffic jams. A vehicle needs a lot of power to accelerate after braking, if this happens a lot, then the fuel-efficiency suffers. Verifying this property is highly dependent on the situation but ensuring that there are possibilities to avoid this behaviour enables the chance to do it. This should encourage self-driving cars to keep the speed as constant as possible. This is also the first and only driving pattern that I consider in the thesis, the other metrics are solely based on sensor values.
The scoring oracle stores the last x performed actions, which can be accelerating or braking. I created a lower limit of 10%, brake or throttle values lower than that are ignored because they barely influence fuel consumption. If the car accelerated the last three seconds, then the list contains three entries that the car pushed the throttle and the corresponding values in percentage. This list gets emptied when the brake and throttle values fall below the 10% limit. However, if a new action is performed, e.g. braking after accelerating the last three seconds, then this counts as an infraction. The penalty score is dependent on the last x values. I assigned this value to four, so the last four actions are considered for the height of the penalty. I take the last x values and calculate an average value, multiply it with ten and round it to an integer, with eight being the maximum. The idea behind this is that the car must have accelerated seconds before and now suddenly brakes. The correct behaviour would be accelerating slowly or keeping its current speed. This works vice versa as well. The same counts for the height of the penalty. The score gets higher because the car, for example, accelerated heavily and now needs to brake. It would be less fuel-inefficient if the AI didn't push the throttle as hard as before.
One example could be the following: the ego-car accelerated for the last five seconds with the values 0.4, 0.6, 0.7, 0.9, 0.8. Now, the test subject brakes with a brake position of 50%, which is over the lower limit of 10%, which counts as an accelerate-and-stop infraction. Because I consider only the last four values, I sum 0.6, 0.7, 0.9 and 0.8, which is 3. The next step is to calculate the average, so $3/4 = 0.75$. Because I consider only Integers between one and nine, I multiply this value with 10 and round it afterwards: $0.75 * 10 = 7.5$ and $round(7.5) = 8$. So the penalty score for this example is eight.

3.3 Test Oracle

In this section, I explain safety-related oracle which validates the driving behaviour of the ego-car at any point of time. I define three states: running, success, failure. The running state means that no test oracle was infracted by the ego-car up to this point. Success means that the success criteria were met and the test case is done. A failed test case means that at least one specification was violated. The simulator provides the test oracles in every iteration the position, direction and velocity of the ego-car, the simulation time and damage sensor states.

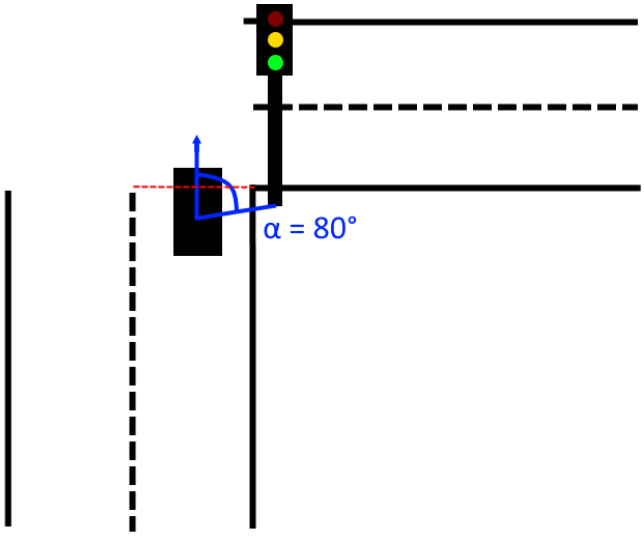

Figure 20: With an angle of 80 degrees between the ego-car and the traffic light, the test subject enters the opposite road. The red dotted line shows where the opposite road starts.

3.3.1 Traffic Rules

Any violation of the traffic rules causes tests to fail. I want to mention that I'm only validating traffic rules for the ego-car. I refer to the traffic rules described in Section 2.3.

Stop Sign
I start counting the simulation time when the test subject is less than ten metres away from the stop sign and its velocity is about zero. I'm calculating the angle between the car and the traffic sign, with an angle of 80 degrees the ego-car is standing partially on the opposite road (Figure 20). If the vehicle under test waits less than three seconds before accelerating and the angle is over 80 degrees, then the test case fails.

Priority Sign
This oracle only applies in the case that no other vehicles drive in front of the ego-car, so there is no reason for the ego-car to stop completely. The test case is failed if the velocity falls to zero and the distance is less than 10 (or twenty for traffic lights) metres to the traffic sign. The distance is experimentally determined. Traffic lights have a higher distance value because they can be placed on multi-lane roads while traffic signs are always on single-lane ones.

Yield Sign
Concerning the yield sign, a car should only stop at yield signs when it's not safe to drive, e.g. approaching traffic on the opposite road, otherwise, the driver is allowed to drive. However, this is a complex task and can be very error-prone to false positives or false negatives. That's why I chose the easy way and force the ego-car to stop in front of yield signs because there is always traffic on the opposite road.

Red Traffic Light
I defined the specifications for traffic lights in Section 3.1.7. Therefore, the test oracle knows the state of the traffic lights at any point in time by validating whether the ego-car entered the trigger point or not. So, if the state is red, the distance to the pole is lower than 20 metres and the test subject passes by the traffic light with a velocity bigger than zero, then the test case fails.

Green Traffic Light
This oracle also only applies in the case that no other vehicles drive in front of the ego-car. In contrast to the red traffic light, if the state is green, the distance to the traffic light is lower than 20 metres and the ego-car stands for two seconds, then the test case fails. The reason why I additionally consider two seconds of standing is because, after the traffic light switches from red to green, I want to give the ego-car a chance to detect the green light and make a prediction.

3.3.2 Damage

I assert that the simulation provides information about the condition of vehicles. So, if a car takes damage and it's listed in the criteria file, the test case fails.

3.3.3 Timeout

Timeouts happen when the simulation time exceeds a certain limit, e.g. three minutes. This value is high enough to say that the test case should have passed in that time, if not, then something went wrong. Because simulations are often executed automatically, this helps users to continue the work without them checking the state of the test case. I set the limit to three minutes, on average my test cases require just above one minute to execute. If the simulation time exceeds this limit, the test case fails.

3.3.4 Success State

A test case succeeds when a traffic participant reaches the intended goal location. This can be any participant and there can be multiple success points. In this case, I define only one point which must be entered by the ego-car. Each point has also a tolerance which can be imagined as a sphere around the centre, the higher the tolerance the bigger this sphere. I compute the euclidean distance between the ego-car and the success point. If the distance is smaller than the tolerance, then the test case was successful.

3.4 Summary

In this section, I described the algorithms and specifications of my test generator, scoring function and test oracles.
The test generator creates urban-like scenarios by incrementally adding either intersections or road segments. The roads are defined by points and properties like width, the number of left

and right lanes as well as type. My validity functions check whether a road network stays valid after adding new pieces. Interpolation creates out of a small number of input points roads with smooth curvatures. After the road generation is done, the test generator adds traffic lights and traffic signs to intersections. Traffic lights have the three modes off, flashing and manual, while the manual mode has two light sequence implementations. Two types of non-ego-cars get added to the test case as well. Non-ego-car 1 spawns and drives only at intersections while non-ego-car 2 drives between two intersections. The last feature adds parked cars, which have three alignment options and they can be placed on both roadsides, on one or none.
The scoring function has five fuel-specific test oracles. The RPM oracle validates whether the ego-car infracts the RPM specification by exceeding the upper limit. The throttle oracle checks whether the test subject pushes the gas pedal too hard by validating the throttle position. A brake infraction happens when the current brake position exceeds the upper limit. The ego-car infracts the engine idle oracle when the engine is running while standing. The accelerate-and-stop oracle validates a driving behaviour pattern. The idea is to validate whether the ego-car can keep constant speed or not. For example, when the ego-car suddenly brakes after accelerating before, then the penalty score is calculated by checking the last x actions (accelerate or brake) and their values. The scoring function calculates a score which assesses the driving behaviour concerning the fuel-inefficiency of self-driving cars.
Test oracles validate the general driving behaviour of autonomous cars. They determine whether a test case fails or succeeds. The first oracle validates traffic rules according to the official German traffic regulations [10] like stopping at red lights and stop signs, yield for other traffic participants at yield signs or keep driving at green lights and priority signs. Another oracle validates for any damage a car takes during test execution. The timeout oracle checks whether the simulation time exceeded the limit or not. Success states mark a test case as successful when a traffic participant reaches the intended goal location.
The next step is to evaluate the functionality and quality of my test generator and scoring oracles.

4 Evaluation

Hereinafter I evaluated how well the various driving behaviours impact the number of infractions and the score. Moreover, I assessed the correlation between my scoring function for fuel-inefficient driving behaviour of test subjects with the consumed fuel, which is provided by the underlying simulator, after test execution. And finally, I measured how effective my generated test cases in regards to traffic lights are. For these goals, I investigated the following research questions:
RQ1: Do various driving behaviours impact the number of infractions as well as the score?
I'm interested to see whether the expected results of specific driving behaviours can be observed in the score and the total number of infractions.
RQ2: Does my scoring function correlate with the consumed fuel?
I want to find out how well the scoring function correlates with fuel consumption. In theory, a higher score means that the test subject consumed more fuel.
RQ3: Can my test generator create effective tests with traffic lights?
Since my test generator places traffic lights automatically, I'm interested whether my test cases can reveal faults in traffic light detection systems.

4.1 Experimental Settings

4.1.1 Driving Simulator

In my experiments, I used the state of the art driving simulator BeamNG.research [5]. Its game engine offers realistic driving physics and accurate sensor measurements. Furthermore, BeamNG.research provides a feature-rich Python and Lua API which covers many user needs, for example, the API provides methods to generate roads inside the simulator or to deploy traffic participants. Also, this simulator is available for free. A feature which comes in handy is the realistic calculation of fuel consumption. According to the official forum of the simulator [20], the BeamNG.research is using many physical processes to approximate the fuel consumption as accurately as possible. Hence, I relied on BeamNG's calculation for fuel consumption in my experiments.

4.1.2 Test Subjects

I used the BeamNG.AI as the main test subject in my experiments, because it is already included in the simulator and it can be configured to simulate different driving behaviours (cautious driver, reckless driver, etc). Even though it has perfect knowledge about the virtual roads and can calculate the optimal driving trajectory, I had to alter its behaviour by setting the waypoints and the speed between each waypoint manually. The reason for this is the AI couldn't stay in the centre of the lane where it's supposed to drive. It either stayed in the middle of the road or arbitrarily changed the lanes. I suppose the AI wasn't build to stay on one lane of a multiple lane road. Therefore, the set speed isn't perfect and the car is driving sometimes too fast on sharp turns. However, the AI offers some tweakable parameters like speed limit or aggression level, which I made use of in the experiments. In particular, this aggression value controls how risky the car drives, e.g. high aggression results in cutting corners, constantly pushing the throttle and brake through and generally driving as fast as possible.

4.1.3 Test Cases

Test Cases for Fuel-inefficiency Evaluation
To answer the first two research questions, I manually created two test cases since they don't rely on automatic test generation. The first test case is called "curvy". The environment only consists of a single road with sharp turns and segments where the test subject can gain high speed. The idea behind this test case was to check whether test subjects can keep a constant speed on straight segments and how they behave in sharp turns. The goal was to stress the subjects to drive with a high RPM on straight segments, accelerating aggressively after a turn, full braking before a turn and combining these to result in an accelerate-and-stop behaviour. The second test case is called "intersection". It is made out of two roads intersecting with each other in a 90 degrees angle. For each direction, I placed a traffic light. The ego-car faces a green light which turns to red when the test subject approaches the intersection, which forces the ego-car to stop. After a few seconds, the light switches to green again, forcing the car to drive. The test case is failed, if the test subject keeps standing on a green light or driving through the red light. Both test cases are plotted in Figure 21. Also, I set the time of day for both test cases to zero (brightest time of day).

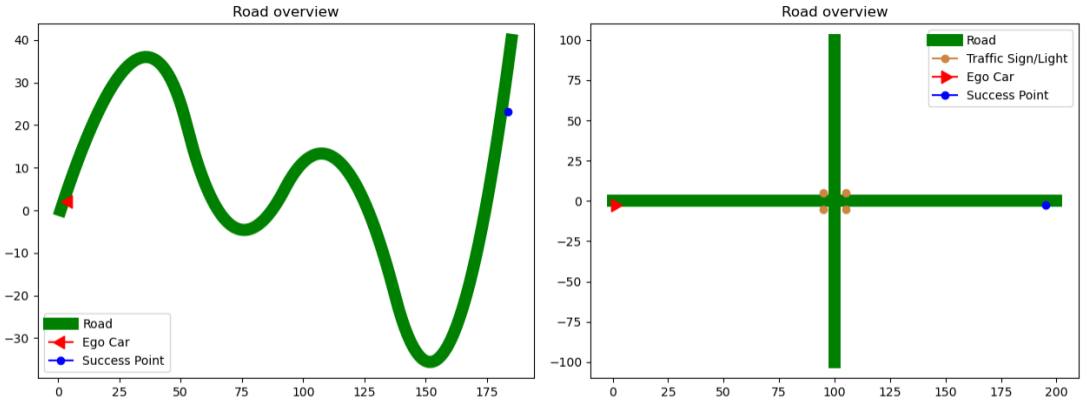

Figure 21: Test case curvy(left) and intersection(right) for the fuel-inefficiency experiments.

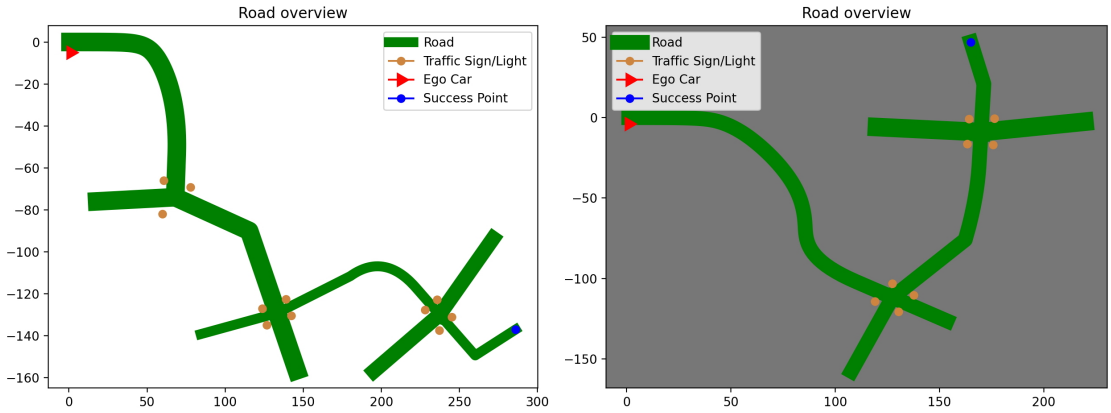

Figure 22: Two test cases of the traffic lights evaluation.

Test Cases for Traffic Lights Evaluation

For this research question, I randomly created eleven test cases with my test generator, but I excluded traffic to focus on traffic lights. The test cases vary in time of day, the shape of roads, the number of traffic lights and the traffic light's initial state. Figure 22 shows two of the eleven test cases to illustrate how significantly they differ from one another. The darkness of the background corresponds to the time of day. A lighter background means that the environment is brighter, a darker means the opposite. I report all eleven test cases in the Appendix (Section 8.2). They are the baseline to all the experiments described in the following sections. In total, four test cases take place in daylight, three at sunset/sunrise and four during the night. All of them have a clear sky.

4.2 RQ1: Do Various Driving Behaviours Impact the Number of Infractions as well as the Score?

4.2.1 AI Profiles

To answer Research Question 1, I created three driving profiles: Aggressive, Default and Role Model. The idea is to make the AI driving in very different ways, and check whether or not differences in the way an AI drives can be captured by utilizing my scoring oracles.
The *Default* profile is the default BeamNG.research AI. This AI is driving a little bit too aggressive (more aggressive than the average driver) and even sometimes cuts the turns. It can keep a constant speed and can stay on the correct lane most of the time. The default script always shifts one gear up when the RPM is near the engine's limit.
The *Aggressive* driver cuts turns most of the time and therefore drifts all the time. This driver always pushes the throttle or the brake to the maximum and always tries to exceed the speed limit (for all test cases set to 50 km/h) and hence, it keeps accelerating and braking during the whole test execution. To achieve that, I increased all aggression variables defined by the BeamNG.AI, including the acceleration while turning, the throttle input and doubled the steering input so the car doesn't drive off the road while taking turns.
The *Role Model* driver tries to minimize all infractions. It shifts up the gears as soon as possible, keeps constant speed, never pushes the throttle too hard, never cuts the turns and drives in general very carefully. For this profile, I decreased the aggression values, the throttle input and manually shifts the gears at certain RPM values. The code checks every second for the RPM value and decides whether to shift the gear or not. I chose one second due to stability reasons, a lower value results in constantly shifting the gear higher and lower. Sadly, I couldn't find a way to turn off the engine when the car is standing still.

4.2.2 Setup

Each AI profile executes the same test case ten times. This should be enough to get stable averaged results. During test execution, my test oracles keep validating the status of the test case. E.g. if the ego car stops at a green light or drives through a red light then the oracle returns the status "failed". Moreover, the scoring oracle collects sensor values every second to validate the fuel-inefficiency specifications.

4.2.3 Results

Infractions and Scores
The results for both test cases are shown in Figure 23. For the test case curvy, the results meet the expectations. The Aggressive driver has the most infractions for all specifications. The biggest parts are RPM and throttle infractions. This is because I increased the throttle input which automatically puts the RPM value to a high level all the time. The brake infractions are also higher since the driver brakes as late as possible and hence, needs to push it harder to stay on the road. Because the aggression level is high, the driver wants to accelerate as soon as possible which leads to a pronounced accelerate-and-stop behaviour in turns, where it keeps braking and re-accelerating.

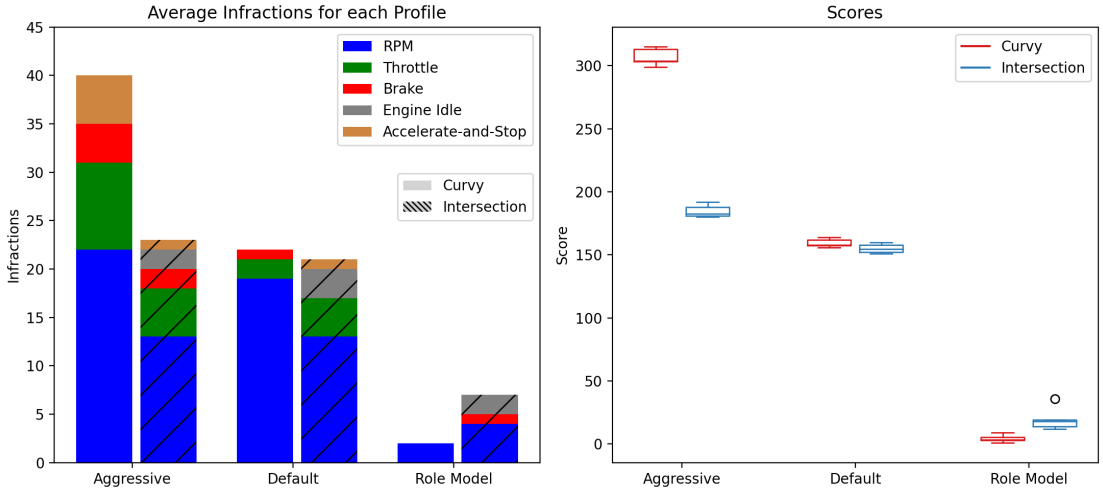

Figure 23: Average number of infractions(left) and scores(right) for both test cases.

The Default AI profile has almost the same amount of RPM infractions since both profiles shift the gear as late as possible. However, other infractions are significantly lower than the Aggressive driver's ones. Nevertheless, because the default aggression level is a little bit too high, other infractions have been spotted.
The Role Model has only some RPM infractions. These could be avoided as well but since the code is checking for the RPM value every second, there are moments in between where the value is exceeding the limit a little bit. Hence, it is spotted as an infraction.
The scores are almost directly proportional to the number of infractions. Sure, the score is dependent on how bad the infractions are but by looking at the plots, the values appear to be on the same level.
To summarize, the Aggressive driver has the highest score, followed by the Default profile who has about 45% lower score and the lowest one has the Role Model with only about 5% score of what the Aggressive driver achieved.
For the test case intersection, the Role Model could again achieve the best results. The Default and the Aggressive profile got the same amount of RPM infractions and almost share the same number of throttle infractions. As expected, the Aggressive driver achieved the highest number of accelerate-and-stop infractions because it can't keep a constant speed on the straight segments. However, all test subjects should have shared the same number of engine idle infractions but the Default profile got the most in this case. One reason why the Role Model has a brake infraction and the Default profile not is because I didn't change the brake input which allows the AI to push it as hard as it wants to. While approaching the intersection both cars pushed the brake too hard, but since the scoring oracle is checking for infractions every second, the Default profile could get away with it while the Role Model got spotted. As explained in Section 4.2.1, I couldn't find a way to turn off the engine automatically, hence, it got some engine idling infractions.
In this test case, the gap of the scores between the Aggressive driver's and the default AI's one is not big. Only the Role Model achieved an outstandingly low score. The scores and the number of average infractions can be seen as directly proportional in this test case as well, even though it was a bit better in the previous one.

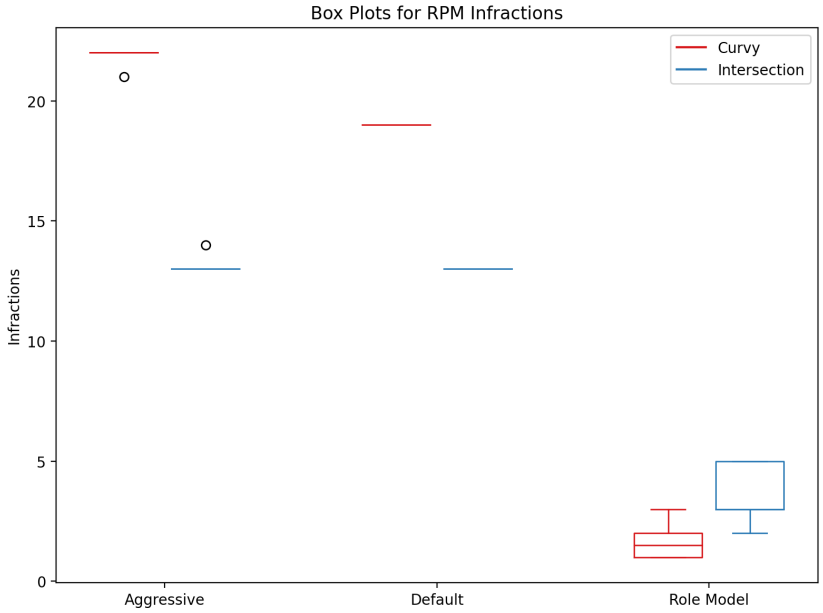

Figure 24: Box plots of RPM infractions of all profiles in both test cases.

How reliable is each infraction?

Figure 24 shows the box plots of RPM infractions for both test cases. For the Aggressive and Default profile, the boxplots are flat lines. The first one has one outlier per test case. For the Role Model, the biggest difference between the minimum and maximum is three, showing that in this case, the RPM infraction is more sensitive. The increased sensitivity comes from checking the RPM once a second, so the value can be over the threshold sometimes because first the RPM is validated and then the algorithm checks whether it should shift or not. Overall, the RPM infraction metric is reliable with a low standard deviation for two out of three profiles, but more sensitive for the Role Model.

For throttle infractions, Figure 25 shows that in five out of six cases, the box plots are flat lines and that the standard deviation is generally low. The only exception is the Aggressive driver in the test case curvy. The median here is about 8.5 and the values lie between eight and ten. This is due to increasing the throttle input. But in general, this metric is reliable.

The box plots for brake infractions in Figure 26 show varying results. In test case curvy, the Aggressive driver and the Role Model share a difference between minimum and maximum of one. This value is low enough and the box plots for this test case prove a low standard deviation. However, for test case intersection, the plots show some outliers for every profile. The biggest attention is drawn by the Aggressive driver: The values range between one and four. The Default profile has only one outlier with a difference to the median of one. The Role Model has outliers plus and minus one to the median. All three profiles have a flat line concentrated on one value. All in all, the results show that the brake infraction metric is sensitive, but can still be considered as reliable since the box sizes are either small or the boxes are flat lines.

Since there is no reason to stop completely in the test case curvy, there won't be any engine idling infractions. In test case intersection, the Aggressive and Default driver have the same plots, the box is a line and both have an outlier with plus one to the median. The Role Model, however, varies between one and three. I argue that this happened by mistake because none of

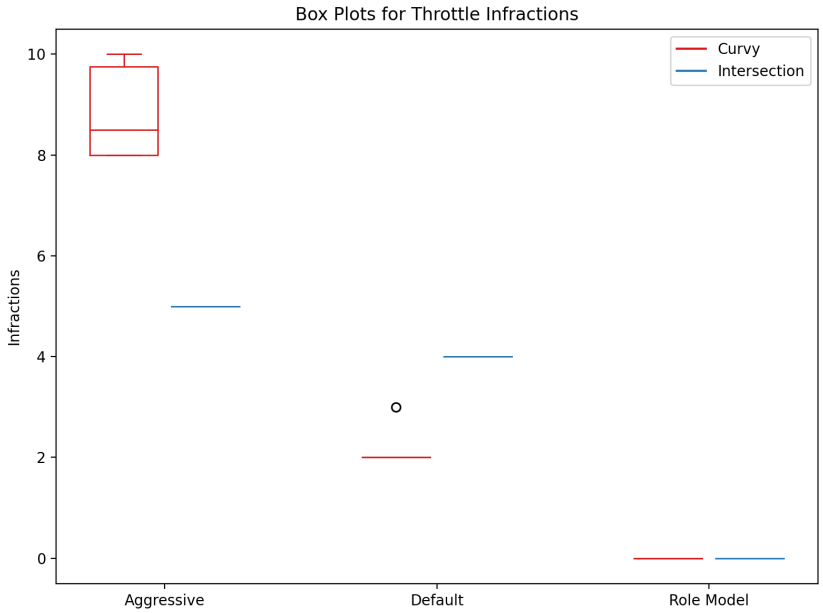

Figure 25: Box plots of throttle infractions of all profiles in both test cases.

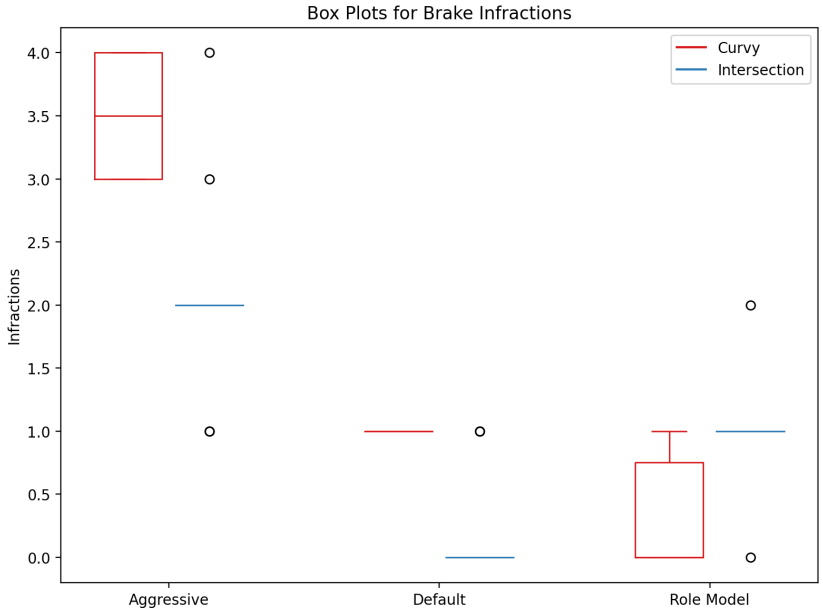

Figure 26: Box plots of brake infractions of all profiles in both test cases.

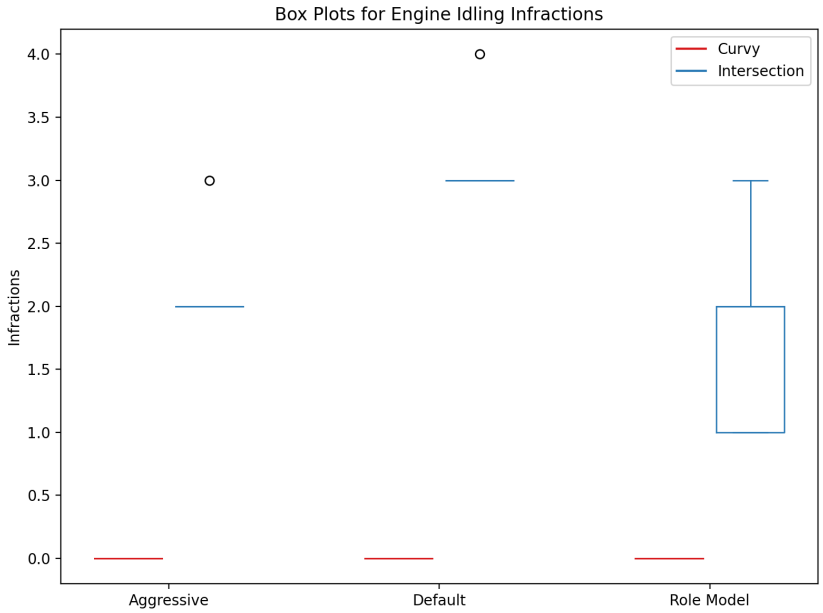

Figure 27: Box plots of engine idling infractions of all profiles in both test cases.

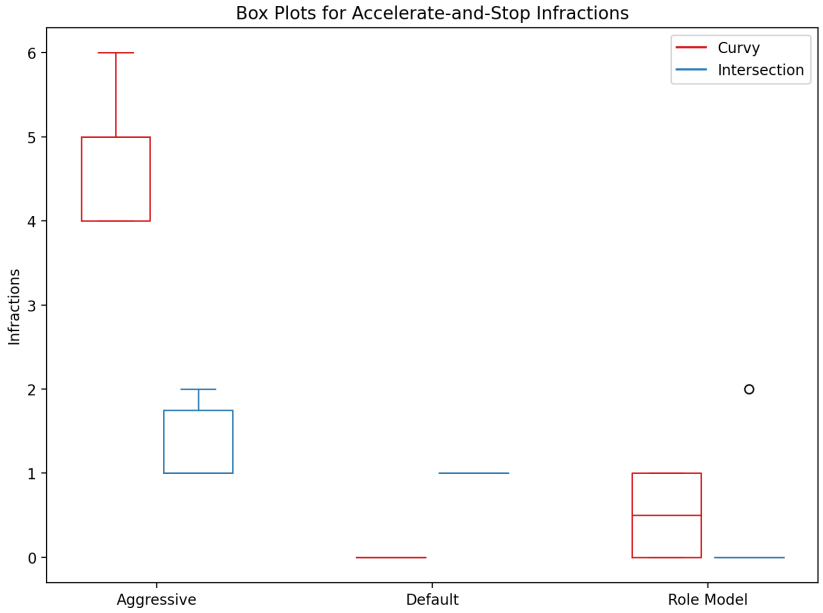

Figure 28: Box plots of accelerate-and-stop infractions of all profiles in both test cases.

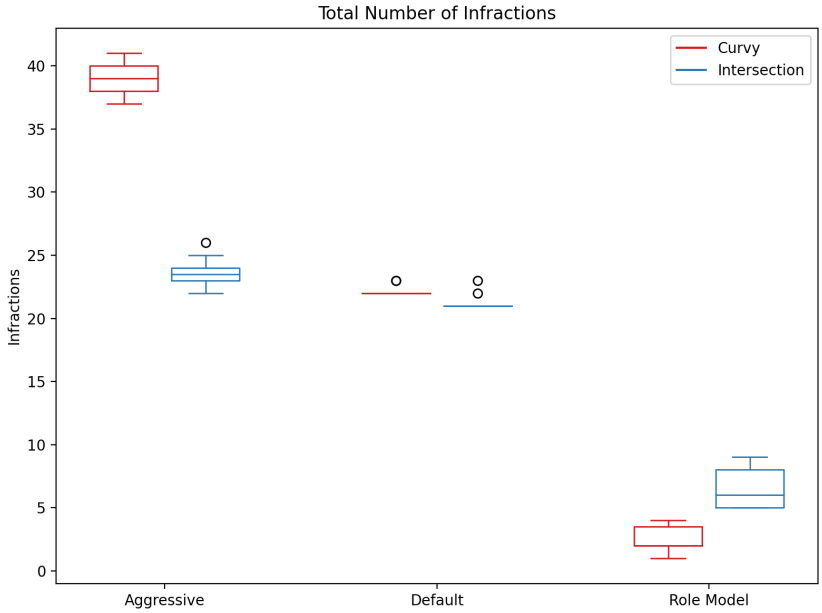

Figure 29: Box plots of all infractions of all profiles in both test cases.

the profiles turns off the engine at intersections. The results are presented in Figure 27. Hence, the engine idling infraction metric is reliable, showing a very small standard deviation for the first two profiles and a bigger one for the last one.
Figure 28 shows that the accelerate-and-stop infraction metric is a bit sensitive to the Aggressive driver. The size of the box plus whisker is two, for the second test case, the total size is one. This shows that the number of infractions mainly vary between two values. The Default profile has two flat lines, showing no standard deviation at all. The Role Model has one flat line and one box which shows that the values are distributed between zero and one. So, the accelerate-and-stop infraction metric is reliable to the two later profiles and is sensitive, even though not too much, to the Aggressive driver.
Generally speaking, the variation between the number of infractions comes from different observation moments. For example, in the first run, the scoring oracle checked the states of the simulation at the timestamp 10:00, in the second run at 10:30. Even though there are only 30 milliseconds apart, the vehicle stats can be different and hence, a different amount of infractions can be observed.
To summarize all results, all metrics can be considered as reliable. The values usually vary between two values, only in two cases they vary between three. Most of the time, the boxes are just flat lines. Exceptions occur for specific profiles in specific test cases, e.g. RPM infraction box plot for the Role Model in test case intersection. The box plots of the total number of infractions in Figure 29 also prove that the values lie within a small range, the biggest difference between minimum and maximum is four.

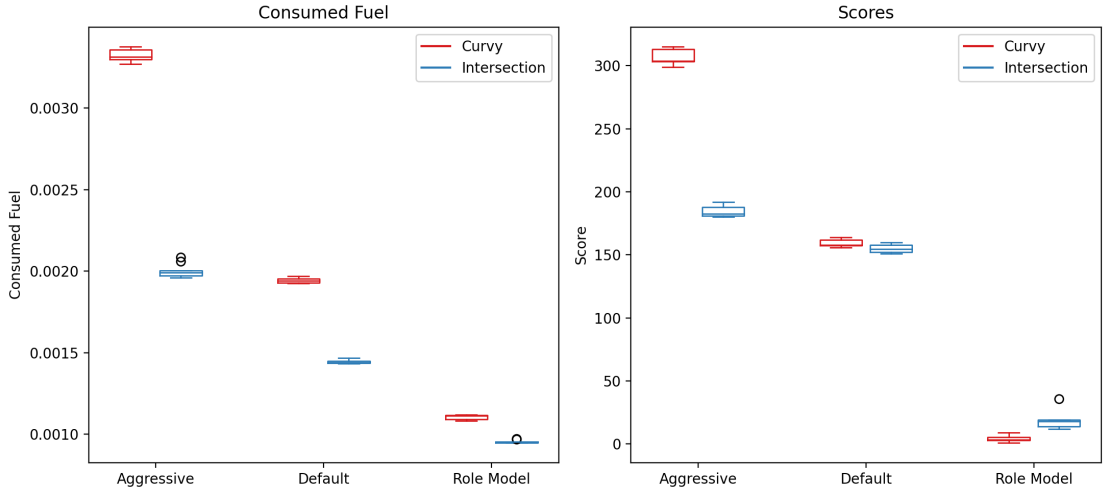

Figure 30: Fuel consumption(left) and scores(right) for both test cases.

Summary
To summarize, the Aggressive driver caused the highest amount of infractions and score, the default AI was in test case curvy substantially better but in test case intersection only slightly better, the score and the infraction number of the Role Model were for both test cases significantly lower than from any other profile. To answer the research question:

RQ1: Various driving behaviours effectively impact the number of infractions as well as the score.

4.3 RQ2: Does my Scoring Function Correlate with the Consumed Fuel?

The setup and the experimental execution are the same as in research question one.

4.3.1 Results

Figure 30 shows the boxplots of the consumed fuel and scores for all profiles for both test cases. For test case curvy, the Aggressive driver consumed the most fuel, followed up by the Default driver and the Role Model consumed the least fuel. One cannot say that a certain score consumed x amount of fuel, that's not what the scoring function was made for. Instead, my scoring function assesses fuel-inefficient driving behaviour of self-driving cars, so the higher the score the more fuel-inefficient it drives. If one compares the Aggressive driver's average score, which is about 310, with the Role Model's average score (about 5), then this would make a difference of about 98.4%. The difference in consumed fuel is about 65.3%. Another reason for this is that each test subject needs a certain amount of fuel to reach the success point, which needs to be subtracted off the results. The problem is that this requires complex calculations to predict this

amount of fuel correctly. Therefore, by looking at the plots one can conclude: The higher the score, the higher the consumed fuel. By looking at Figure 30, the levels are about the same. Only the Default profile is a bit of an outlier.
The same observation counts for test case intersection. However, in this case, the Role Model has about a five times higher score but consumed less fuel. Also, the Default profile lies somewhere near the Aggressive driver but consumed considerably less fuel. These observations imply that my scoring function is not perfect and needs more experiments to find the right configuration of values. This could be problematic for AIs which don't differ too much in fuel-efficiency. One AI could achieve a higher score but the results show that it consumed less fuel. But for extreme test cases like these three profiles, the scoring function shows which driver consumed less fuel.
To assess the correlation between the consumed fuel and the score quantitatively, I calculated the Kendall correlation of all test executions per test case. There are many correlation functions, two popular ones are the Kendall and Spearman correlation [13]. Both are known to be robust, but I prefer the Kendall correlation over the Spearman one because Croux et al. [13] showed that the first one is more robust, more efficient and has a lower mean squared error. Also, the Kendall correlation is prefered when the dataset is small and consists of many tied ranks, which fits my case. The Kendall correlation for test case curvy is about 0.839, for test case intersection about 0.790 and for both together the value is about 0.779, which means there is a strong positive correlation in both cases between the consumed fuel and the score (just for comparison, the Spearman correlation is 0.957 and 0.937). In other words, I showed quantitatively that the higher the score the higher the consumed fuel of a car-under-test. I separated the correlation for each test case because the consumed fuel is scenario-specific. This means that one profile in different test cases can reveal the same amount of consumed fuel, but the number of infractions is differently distributed. However, I showed that for both test cases the correlation is almost the same.
To summarize, my scoring function can be used as a means to assess the fuel-efficiency of self-driving car software based on their driving behaviour. To answer the research question:

RQ2: The scoring function correlates strongly with the consumed fuel.

4.4 RQ3: Can my Test Generator Create Effective Test Cases with Traffic Lights?

4.4.1 Traffic Light Detection System

To evaluate the traffic lights of my test generator, I installed and configured a traffic light detection system, which I found on GitHub [23]. This detection system was made by a group called "Affinis AI Laboratory" [1] to recognize traffic lights in the open-source driving simulator CARLA [16]. Their dataset for training and evaluating was also collected in CARLA. The detection system can only predict green and red traffic lights. Also, one developer in the GitHub readme file states that images, that are too far away won't have annotations. By looking at the dataset [22], one can see that all images were collected at the same time of day (on daylight) and the traffic lights aren't far away. Also, the sky is cloudy and hence, the background is rather light grey. This causes problems with clear skies at daylight like in Figure 31. To facilitate the task, I used the pre-trained model that the developers provide. The input is a recorded PNG image and the model stores the same image with a bounding box and the label. I decided to treat the "off" label (disabled traffic lights) the same as no prediction of the detection system.

Figure 31: Background is classified as a green traffic light.

Figure 32: Correct prediction at green traffic at daylight.

Figure 33: Correct prediction at red traffic light at nighttime.

One could say that there are existing detection systems which can spot disabled traffic lights, in this case, there would be a difference to no predictions by the AI because this would be a faulty behaviour. However, I argue that the outcome would be the same for both cases because in either way the test subject must rely on traffic signs.
The Figures 32 and 33 show correct outputs of the detection system on two example images that I recorded.

4.4.2 Traffic Lights Implementation in BeamNG

BeamNG.research does currently not provide any interface to control traffic lights. Only the level "west_coast_usa" provides traffic lights and the way it works there is by shifting a PNG image from left to right, which contains the colours for each light bulb, in a certain amount of time. Therefore, I created a system which teleports real light objects between their position on the traffic light and under the ground, where they can't be seen. To have full control when the lights should switch, I added trigger points which trigger certain behaviour. For example, the traffic light has an initial state of green. 40 meters apart from that I place a trigger point. When the car enters this trigger point, the green light gets teleported under the ground and the yellow light teleports from the invisible area to its correct position. After one second, the yellow light gets teleported under the ground and the red light is set to its correct position. The advantage over using traffic light sequence images is because this enables to trigger certain events, e.g. when the car approaches the intersection then the light is switched to red. With an image, one needs to calculate the time of arrival which can be very difficult. The problem is that this needs to be done in Lua, but the test oracle runs in Python. Even if I copy the same rules to trigger a certain event in Python then this won't detect them reliably. The biggest problem is to get the state

of the traffic lights. For example, the current state of the traffic light is yellow, but in Python, the oracle returns red. This is happening because there is no way to exactly synchronize Python with Lua. For Lua, BeamNG.research provides a function called "onRaceTick", which executes the code block every 250ms. Python executes the code block as soon as every line of code was already executed. Let's look at the worst case: Python registers that the ego car entered a trigger point. In Lua, the entrance was registered 249ms later. Python thinks for 249ms that the traffic light state is yellow while in-game it's green. The light switches from green to yellow and after two seconds in the simulator to red. Again, for 249ms the test oracle in Python thinks the state is red while it's yellow. This goes on until the traffic light reaches its end state. Therefore, I ignore certain periods between two light switches to get a more reliable prediction of the traffic light state. Otherwise, the output is None, which means undefined.

4.4.3 Setup

For each generated test case, the ego-car drives according to a predefined path and collects images. To avoid images without traffic lights or that are too far away, I save them only when the traffic lights are in the field of view of the ego car and when the distance between the test subject and the traffic light is under 65 meter. I also skip every image that is near the light switching phase to avoid false labels, which is described in Section 4.4.2. This comes with the cost of collecting fewer images, but less wrong labels. But still, out of 1522 images, only three had a false label because I checked every image manually. The resolution of the images is the same as the dataset that is used for training. The names were labelled with the ground truth states of the traffic lights. However, there is a bug in BeamNG.research where the taillights of the ego car are also visible in the image. Hence, many pictures contain two red circles in the centre of the image. They vary in opacity, size and strength. This is a big problem because the traffic light detection system can make false predictions due to the dominance of the red colour. After that, each test case gets evaluated with the detection system. I set the limit of false predictions to six and predict the output of the AI with the ground truth value.

4.4.4 Experiments

To answer the research question, I created six experiments of similar type.

Random
The first one is called "random", where I used the randomly generated test cases without any modifications. The minimum distance between the traffic light to the test subject to save images is set to eight meters and maximum distance to 65 meters. This experiment should test the ability to recognize traffic lights correctly at any given time of day.

Daylight:
In this experiment, I set the time of day for all test cases to zero, which means the brightest environment possible. The dataset which was used to train the detection system contains also only images at daylight and should reveal whether the system performs better at this time or not.

Nighttime:
This is the opposite of the daylight experiment. The time of day for all test cases is set to 0.5

which has the darkest environment. I wanted to compare the results of the daylight experiment with the one in the night to show possible faults of the detection system.

Cloudy:
The setup is the same as the daylight experiment, but the sky is now full of clouds and generally grey. The environment is the same as the dataset for the detection system. The idea was to reduce misclassifications caused by a greenish background.

Max distance 15 meters:
Here, the maximum distance to record an image is set to 15 meters, the same distance that is used by the developers of the traffic light detection system. The sky is cloudy and the time of day is set to zero. These settings are meant to be as close as possible to the training dataset. The minimum distance is set to seven to increase the number of recorded images. In theory, this should have the best results.

Max distance 45 meters:
For this experiment, the maximum distance to record an image is set to 45 meters and the minimum to 16. This should represent a complement to the experiment above and the results are expected to be worse. Again, the sky is cloudy and the time of day is set to zero.

4.4.5 Results

For the previous experiments, I consider test cases which contain six or more images of yellow and yellow-red traffic lights automatically as failed because the detection system that I use is only able to classify the states red and green. So, there are test cases which must fail and test cases which should have passed. That's why I checked every test case for each experiment manually whether it should pass or not. However, I still consider all eleven test cases because even though some always fail, they can reveal issues of the detection system. I also manually checked the validity of the generated test cases and the ego-car can drive from start to finish without violating the specifications of the test oracle. Hence, I consider my test cases as valid and there are no test cases in this experiment which should have failed but were predicted as a success by the detection system. That's why there is always a zero in the confusion matrices in the bottom left corner. Additionally, I don't intend to create such test cases.
First, I talk about the results of each experiment and second, in the summary chapter, I explain what the results tell about my test generator and answer the research question.

Random
According to the confusion matrix in Figure 35, seven test cases should have passed but only one did. There are some reasons why they failed. The first one is because the detection system is only able to classify the states green and red, which raises the number of false predictions per test case for the colours yellow and yellow-red. The second reason is that the background is getting classified as well because it's greenish in bright environments. I discussed this issue already in Section 4.4.1. But even if the traffic light was red and the ego-car was right in front of it, the background dominated and the detection system predicted green. Also, the detection system keeps classifying yellow-red lights as red which can be seen in Figure 37. This makes sense because the system is only able to classify the states red and this traffic light states contains this colour. The fourth reason is that traffic lights which are too far away are barely recognized which is acknowledged by the developers. In test cases during nighttime, the detection system

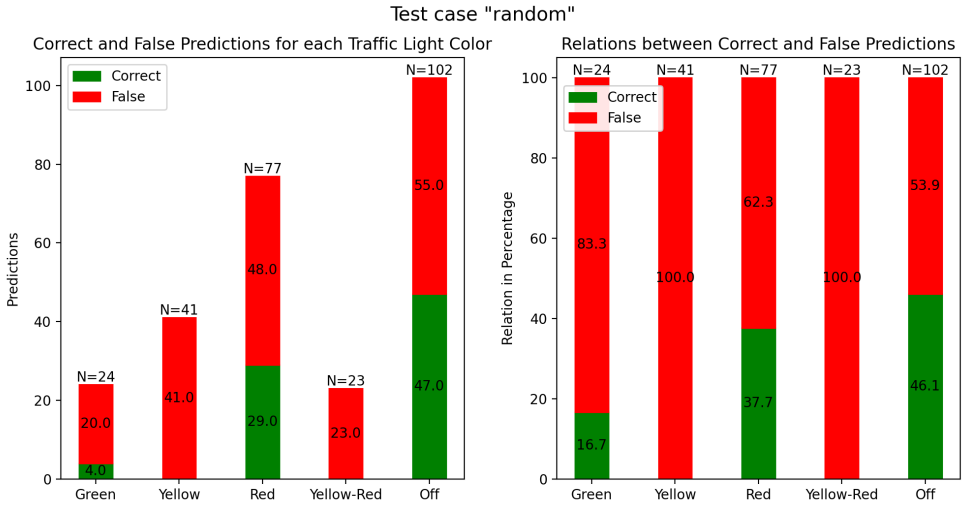

Figure 34: Number of correct and false predictions per traffic light color (left) and relations between them (right) in test case "Random".

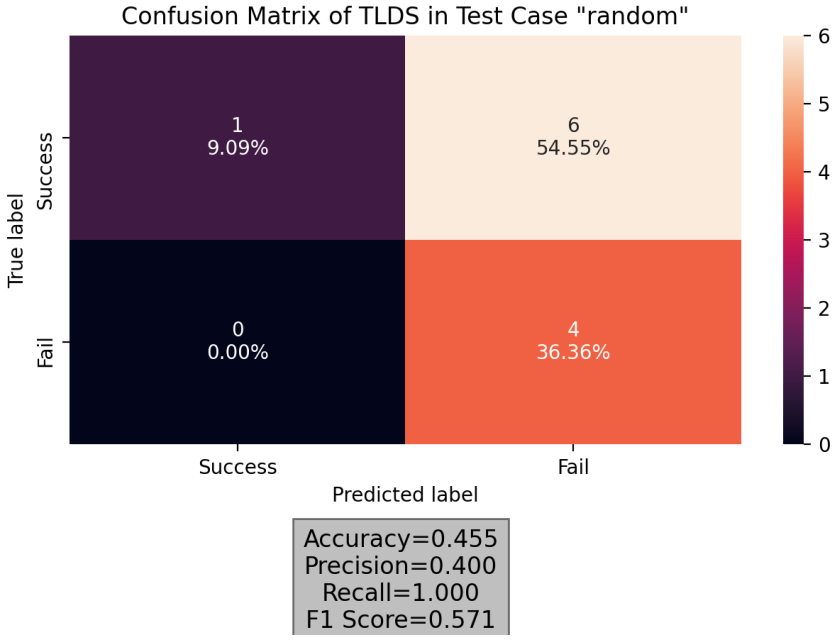

Figure 35: Confusion matrix of the traffic light detection system of the experiment "Random".

Figure 36: Yield sign is predicted as red traffic light.

Figure 37: Yellow-red traffic lights are classified as red.

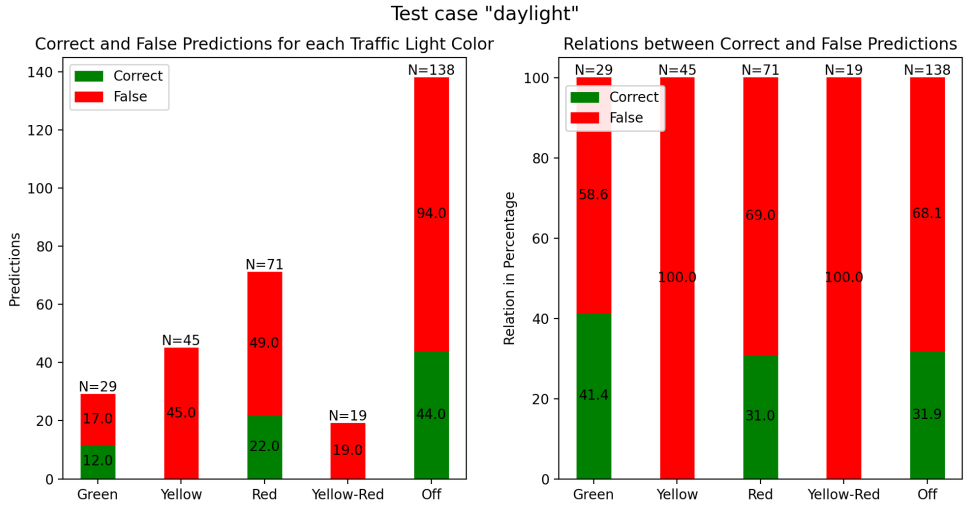

Figure 38: Number of correct and false predictions per traffic light color (left) and relations between them (right) in test case "Daylight".

had problems recognizing some traffic lights even if the ego-car stood right in front of it. This was to be expected because the training dataset consists of only images during the daytime. Another mistake was that the AI sometimes predicted yield signs as red as can be seen in Figure 36.
My test generators could create many different environments, and the tests identified in this test subject various issues. Figure 34 shows the number of correct and false predictions for each traffic light and the relation between them. Barely any green traffic lights were recognized and more than half of disabled traffic lights were classified as green or red.

Daylight
Figure 39 shows that all six test cases, that should have passed, failed. The main reason for this is again the green background which keeps getting classified as a green traffic light. Especially when the traffic lights are off, they are getting labelled as green; the same happens with some red traffic lights as well. Even if the bounding boxes are around the traffic light, the system gets confused with the background colour. The distance is once again problematic. Figure 38 shows that the colour green has a better correct to false prediction ratio than the red one this time. I assume that the background helped a lot in this case, with the cost of a worse ratio for the red colour and disabled traffic lights than in the "random" experiment.

Nighttime
Out of five test cases, which should have passed, all of them failed (Figure 41). The main reason for this is because the detection system did barely make any predictions. Even if the ego-car stood in front of the traffic light and its headlights shined at them, it didn't recognize them many times. Even more problematic were traffic lights that are further away because they had even lower chances to get recognized. However, the detection system could recognize some red traffic lights reliably, the green colour was ignored most of the time. I assume that the red colour is more dominant in dark environments than the green colour. The high ratio of correct predictions of disabled traffic lights (Figure 40) follows my explanation from above and also since

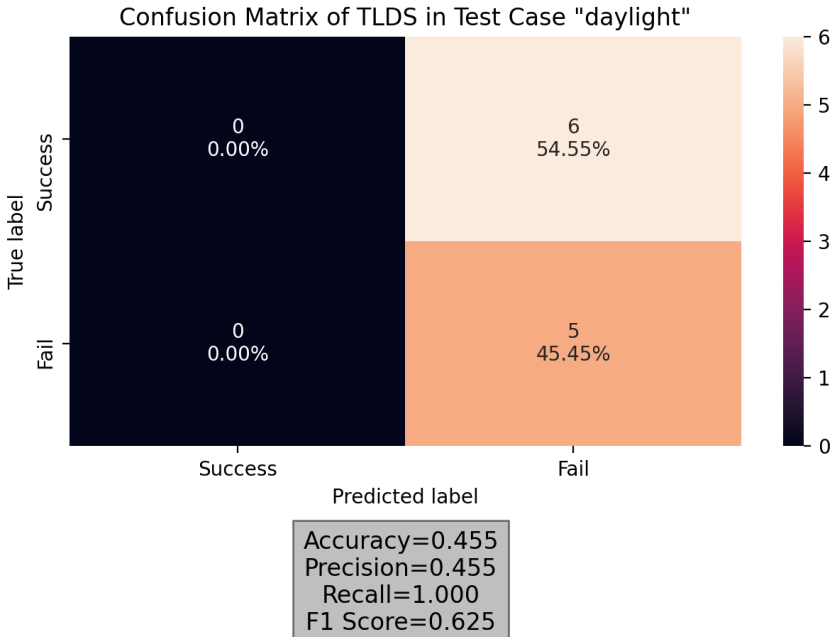

Figure 39: Confusion matrix of the traffic light detection system of the experiment "Daylight".

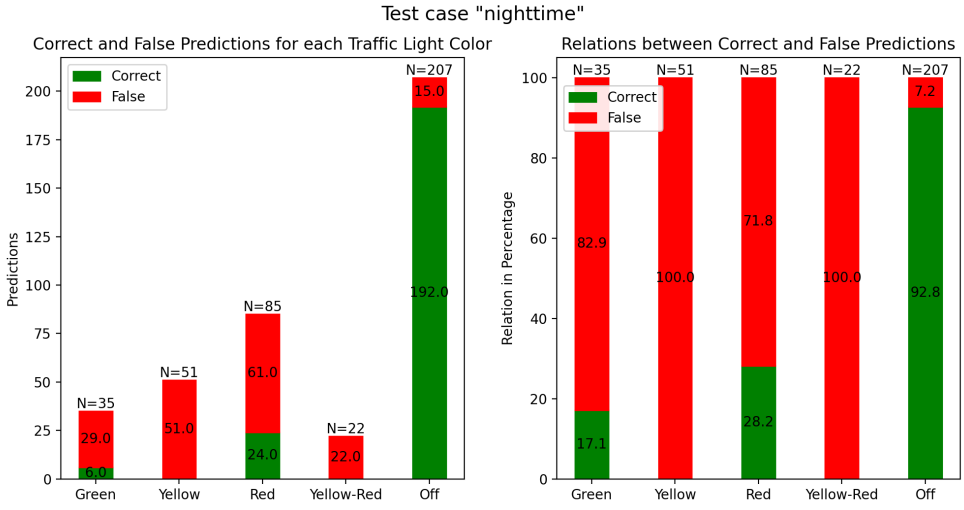

Figure 40: Number of correct and false predictions per traffic light color (left) and relations between them (right) in test case "Nighttime".

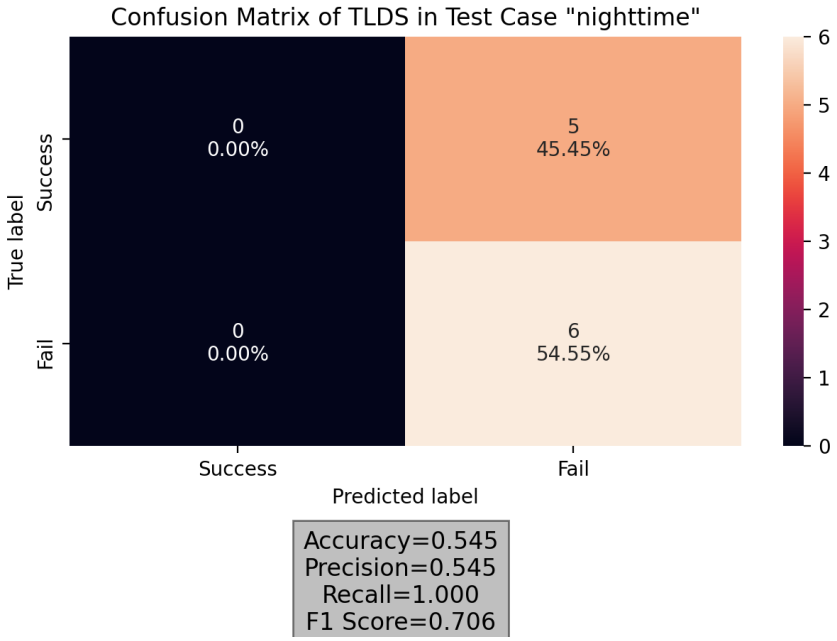

Figure 41: Confusion matrix of the traffic light detection system of the experiment "Nighttime".

the background was very dark, they didn't get confused with the colour green as in the daylight experiment. On the one hand, this experiment shows that the detection system isn't trained to recognize traffic lights during the nighttime and hence, is missing many predictions. But on the other hand, this emphasizes the fact that the colour of the background matters a lot as shown by the different results between this experiment and the daylight one.

Cloudy

In this experiment, one test case out of six succeeded, like Figure 43 shows. Again, traffic lights that are far away are barely recognized. Also, many disabled traffic lights are misclassified as green, the same observations can be made in the daylight experiment. Since the sky is grey for all test cases, this implies that disabled traffic lights are generally bound to be classified as green by the detection system and exposes a faulty behaviour. Some predictions also seem to be arbitrary; red lights are sometimes classified as green, red classifications are more likely to happen when the yellow light is visible, yellow lights are randomly labelled. There are many observable problems in this experiment.
But as expected, according to the results in Figure 42, the detection system performed better than the daylight experiment and could get a better ratio for correct predicted green and red traffic lights than the nighttime one.

Max distance 15 meters

In this experiment, even though the detection system could succeed in the most test cases, it is important to mention that for each test case there were far fewer images than in the other ones, meanwhile, the limit to fail was still set to six wrong predictions. So, the confusion matrix isn't significant in this case. More interesting are the results in Figure 44. Green traffic lights

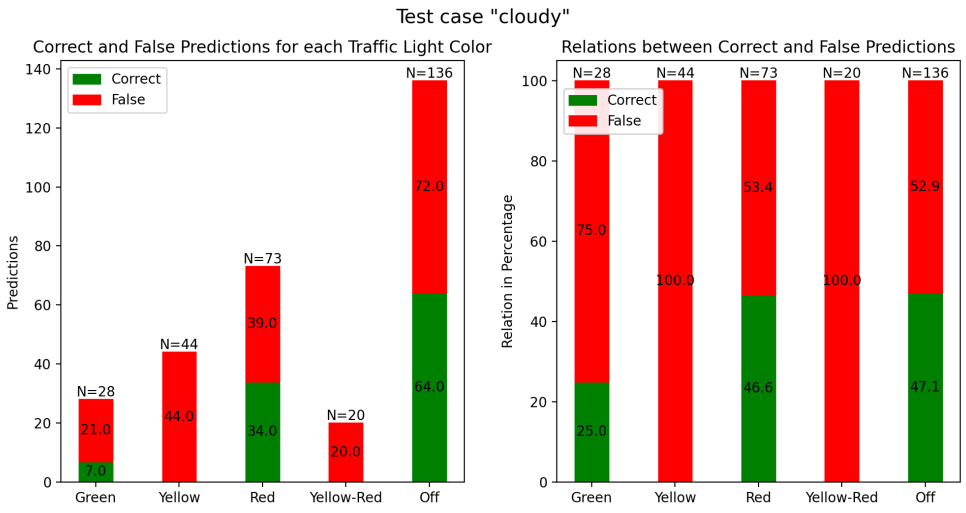

Figure 42: Number of correct and false predictions per traffic light color (left) and relations between them (right) in test case "Cloudy".

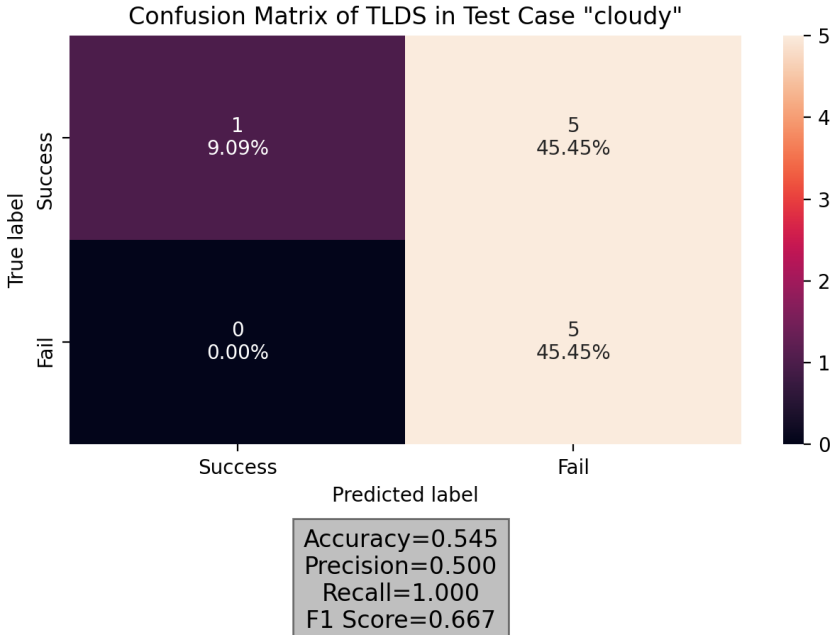

Figure 43: Confusion matrix of the traffic light detection system of the experiment "Cloudy".

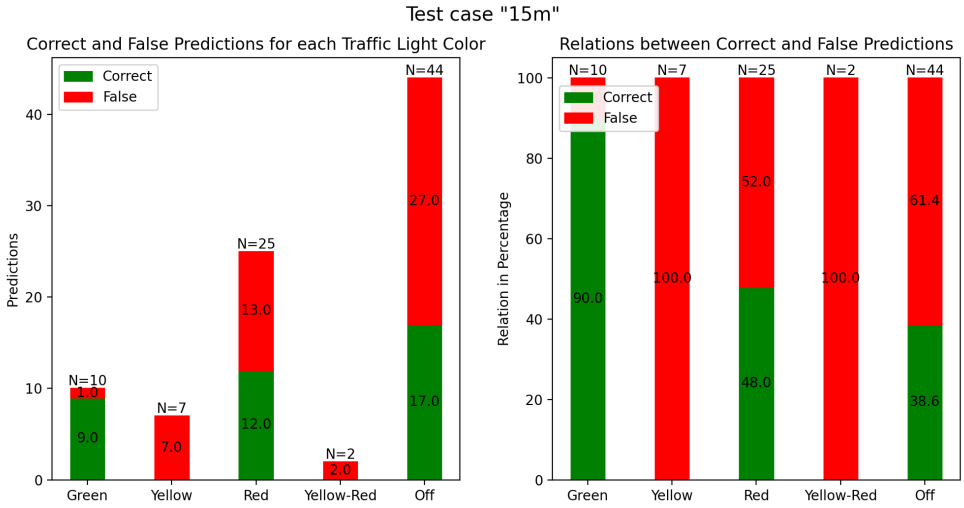

Figure 44: Number of correct and false predictions per traffic light color (left) and relations between them (right) in test case "Max distance 15 meters".

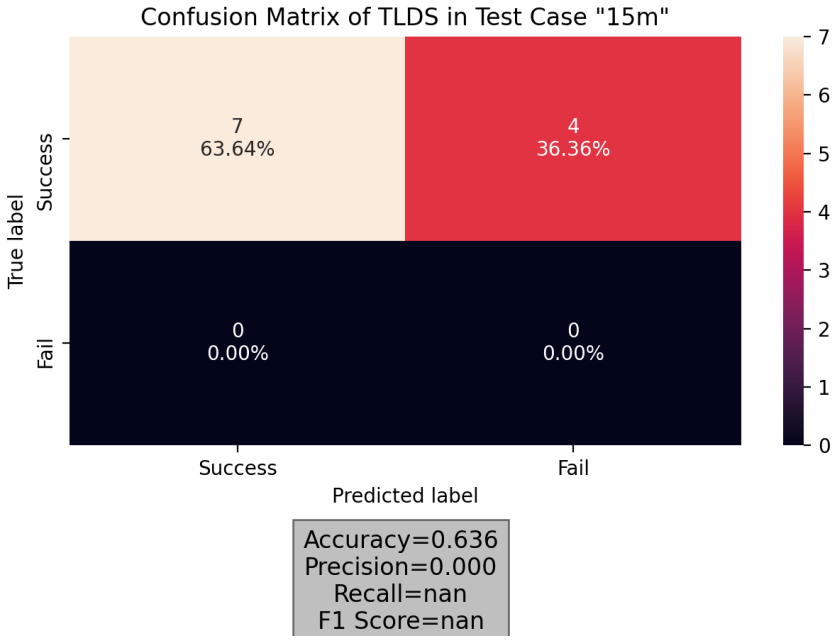

Figure 45: Confusion matrix of the traffic light detection system of the experiment "Max distance 15 meters".

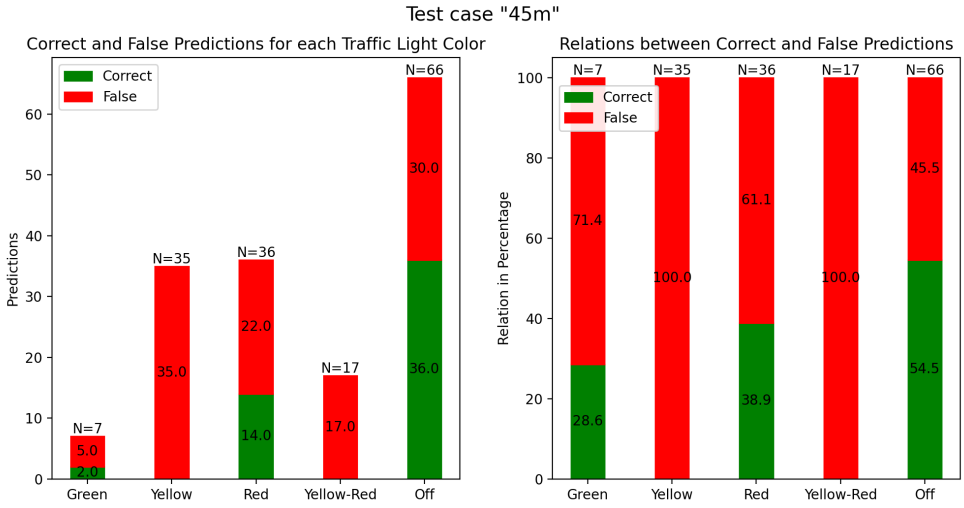

Figure 46: Number of correct and false predictions per traffic light color (left) and relations between them (right) in test case "Max distance 45 meters".

could be predicted correctly most of the time and no experiment has a better ratio for red lights. The "off" ratio is the second-worst because, as in the cloudy experiment, disabled traffic lights are classified as green and sometimes as red because of the yield sign. I could also observe that almost every image has a bounding box, meaning that the detection system performs well with traffic lights which are at most 15 meters away from the driver.

Max distance 45 meters

This experiment also contains fewer images but still only two out of seven test cases passed. Prediction-wise, this one has the same problems as "cloudy" and "Max distance 15 meters". However, it is noticeable that only about half of the images have bounding boxes. This means that increasing the distance between the ego-car and traffic lights by up to 30 meters plays a huge role whether they get recognized or not. Hence, the results are worse than in the "Max distance 15 meters" experiment, except for disabled traffic lights. The ratio of correctly predicted disabled traffic lights was better here because the distance to traffic lights was bigger. Therefore, most of the time the lights were not recognized by the detection system. The results can be seen in Figure 46.

4.5 Summary

Except for the "Max distance 15 meters" one, in every other experiment, the test subject struggled and did not pass many of the generated test cases. I could reveal many faults of the traffic light detection system caused by many different reasons, even if the environment was as similar as possible to the original training dataset. Therefore, my test generator can create test cases which expose various failures of the detection system and to answer the research question:

RQ3: My test generator can create effective tests with traffic lights.

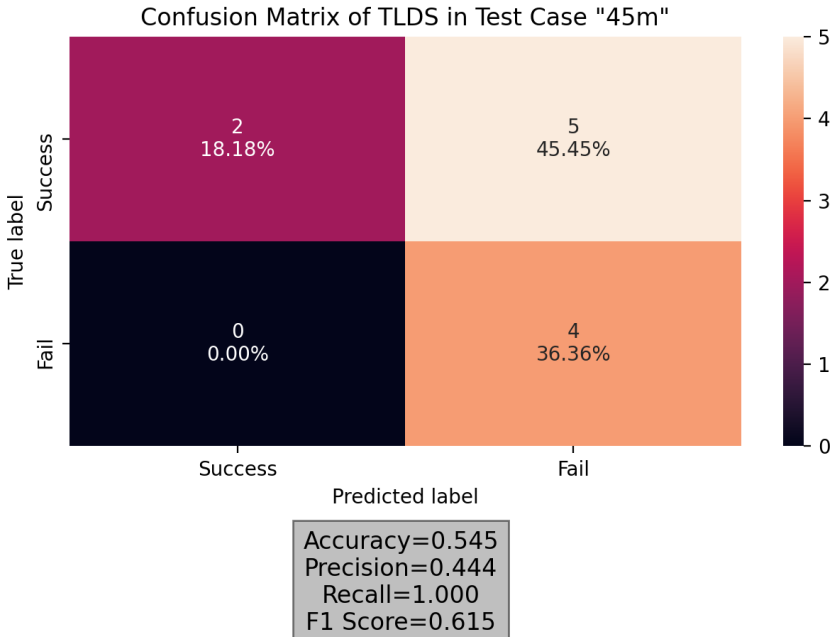

Figure 47: Confusion matrix of the traffic light detection system of the experiment "Max distance 45 meters".

4.6 Qualitative Evaluation of Intersections

Next, I want to qualitatively evaluate the generated intersections. For this purpose, I created screenshots to demonstrate the ability of my test generator to create intersections with different features, geometries and configurations.

Figure 48 shows test case intersection from the fuel-inefficiency evaluation. Instead of using traffic lights, the test generator can easily replace them with traffic signs, e.g. in this case instead of facing red traffic lights the ego-car now faces a stop sign. The driving task stays the same: the test subject must stop at the intersection, otherwise the test case is considered as failed.

The test generator itself can create various kinds of intersections. Figure 49 shows four randomly generated intersections. The number of lanes, the layout, directions, traffic lights and signs can be set arbitrary. This Figure also shows that roads with one lane get a single traffic light while multiple lanes roads get two.

4.7 Qualitative Evaluation of Traffic Lights

As well as the previous sections, I also qualitatively evaluated the traffic lights.
Figure 49 shows all three colours separately plus the off mode. My video (`https://youtu.be/TvZF2vMQ6-c`) [36] shows the flashing mode, the cycle with the initial colour green and red as well as traffic lights during day and nighttime.

Figure 50 shows that single and double traffic lights can be attached with priority and yield signs. This becomes useful when the flashing or disabled mode is active because then the signs on the pole are controlling the traffic.

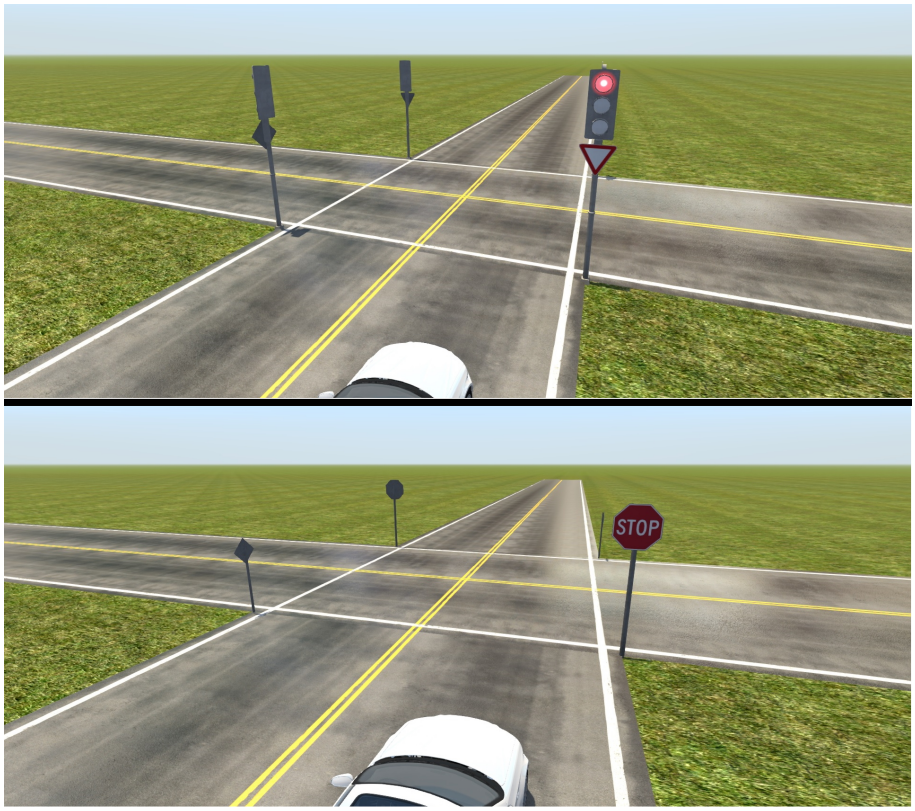

Figure 48: Test case intersection. Top image shows intersection with red lights, bottom image shows the same intersection but with traffic signs.

4.8 Qualitative Evaluation of Traffic Signs

Next, I qualitatively evaluated traffic signs. I'm using only the priority sign (Figure 51) and stop sign (Figure 52) at intersections. The figures show how they look like inside BeamNG.research.

4.9 Qualitative Evaluation of Traffic

Section 3.1.9 showed two kinds of traffic participants, their possible spawning locations and finishing points as well as how they are triggered. To demonstrate this in action, I recorded four videos. In this one (`https://youtu.be/RSk3Vost15k`) [37], the ego-car approaches a three-way intersection and faces a stop sign. The non-ego-car on the opposite road gets the command to drive its route and turns right.
In video (`https://youtu.be/1oURP4ls5b0`) [38], the non-ego-car spawns on the opposite lane and turns left at the intersection, where it traverses to the end of the road.
In the third video (`https://youtu.be/vp10T44xASs`) [39], the car on the opposite lane drives past the ego-car until the road segment ends.
This video (`https://youtu.be/hn-E9AwmJ2o`) [40] presents the second participant type. The car is triggered as soon as the simulation starts and just passes by the ego-car until it reaches the end of the road.

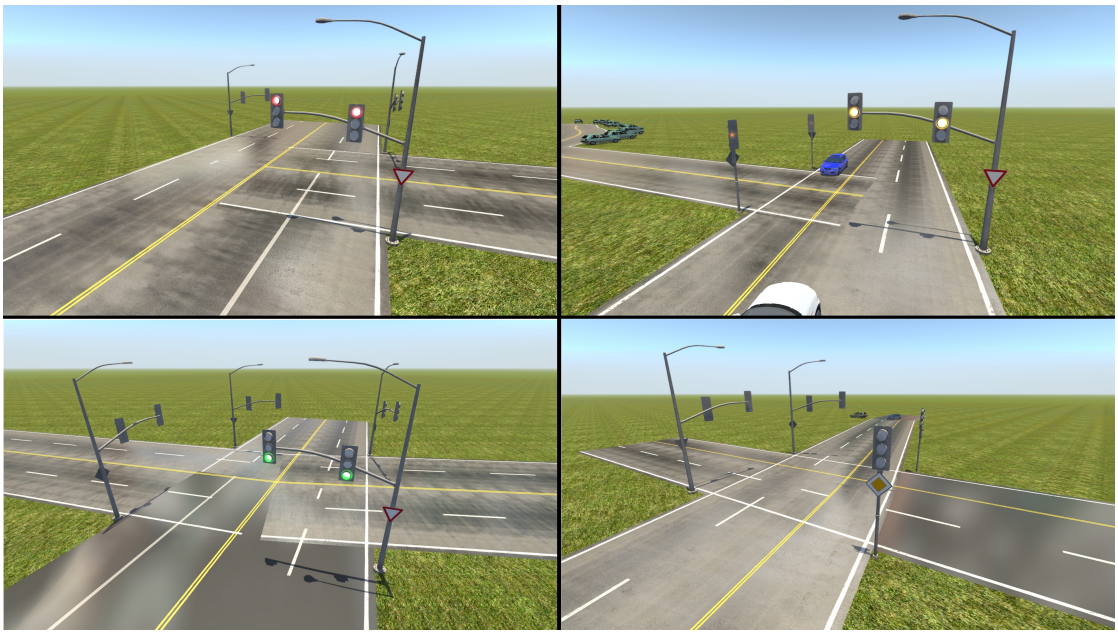

Figure 49: Four intersections from arbitrary generated test cases.

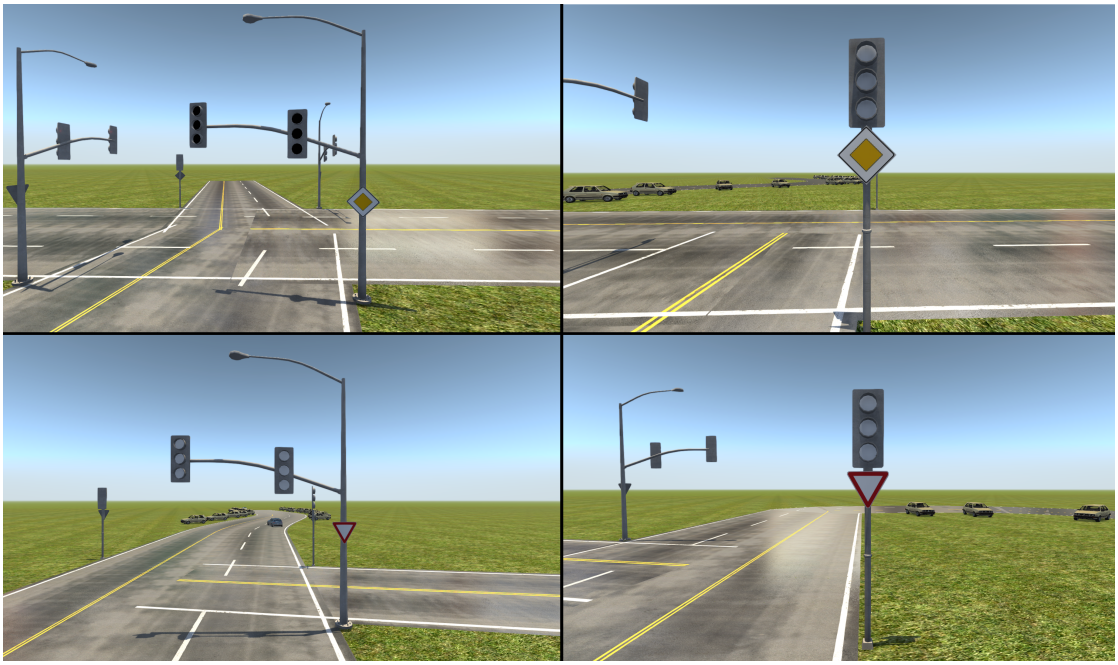

Figure 50: Single and double traffic lights with priority and yield signs attached.

Figure 51: Priority sign at an intersection.

Figure 52: Stop sign at an intersection.

4.10 Qualitative Evaluation of Parked Cars

Section 3.1.10 shows the various possibilities to place parked cars into a test case. In this chapter, I want to demonstrate how the generated results look like inside BeamNG.research.
Before that, I must clarify one important thing. BeamNG.research has vehicle objects and static objects. They have a lot of differences. Vehicle objects can drive, have soft-body physics and hence, can be damaged, have their methods implemented in the Python API and inside the game engine and can be modified separately. Static objects have no physics and the same object class share the same configuration. The simulator must keep track of the physics of all vehicle class objects and therefore, even high-end CPUs can't handle many vehicles. I have performed experiments to see how many cars my Intel Core i5-8400 CPU with 6 cores and 2.8GHz can handle. For six vehicle objects, the CPU workload already hit 100%, the loading time took about 30 seconds longer, but still had steady 45 frames per second. For eight vehicles, the frames per second number dropped to ten. For ten vehicle objects, the loading time increased by one minute and I had only three frames per second. 12 vehicles crashed BeamNG.research and my computer froze. This led me to choose static objects to represent parked cars. On the positive side, the loading time and the frames per second were not noticeably affected by adding parked cars as static objects. On the negative side, the cars share all the same colour, even though my test generator can to assign each car its own. I couldn't find a way to overcome this problem. One possibility would be to copy the car model x amount of times, but since it takes about 100MB of storage, this isn't a profitable solution. Maybe in future versions of BeamNG.research, this would be possible.
Also, I couldn't take an existing car model from BeamNG.research because they have a special way to load vehicle objects into the simulation. The baseline is a DAE file, which contains the shape and material of the car. To be mod-friendly, the developers decided to use JBeams to modify the car easily, e.g. one could apply another skin, a different engine or wheels to the same car model [4]. However, BeamNG uses DAE files for static objects and no JBeam files. Luckily, I found a car model in the BeamNG forums which uses only one DAE file to define a vehicle object, which can be used as a static object as well. It is a 1987 Volkswagen Golf MK2 replica made by the user "RedBolide" [58] and can be seen in Figure 53.
Figure 54 shows four examples of how parked cars can be placed. They vary in degrees to the road, missing spots, distance to each other, distance to road, colour and side of the road.

4.11 Qualitative Evaluation of Test Oracles

In this section, I want to qualitatively evaluate my test oracles by presenting the functionality in videos. As I already described in Section 3.3, the test oracle validates some traffic rules at intersections. In this video (`https://youtu.be/P3ckKR6dVgY`) [30], the test subject doesn't stop at a red light and hence, the case is failed and the simulator closes. The same rules apply for stop signs, as can be seen in this video (`https://youtu.be/NBtvV9IrGbo`) [31].
For green traffic lights, the test subject is not allowed to stop completely. This video (`https://youtu.be/t8OsyUgnUtE`) [32] shows that the test oracle can spot this infraction and stops the execution. The same applies to priority signs as well (`https://youtu.be/6Mcu6wVkBlA`) [33].
Video (`https://youtu.be/kiD8AYsFKpI`) [29] shows that the test oracle also detects crashes.
To show a complement to failing test cases, I've generated two random test cases showing all features where the ego-car can successfully reach the destination point. They can be watched here: (`https://youtu.be/2y6nYWOawx8`) [34] and (`https://youtu.be/9cJ5yX68FtQ`) [35].

Figure 53: The Volkswagen Golf MK2 replica made by the user "RedBolide".

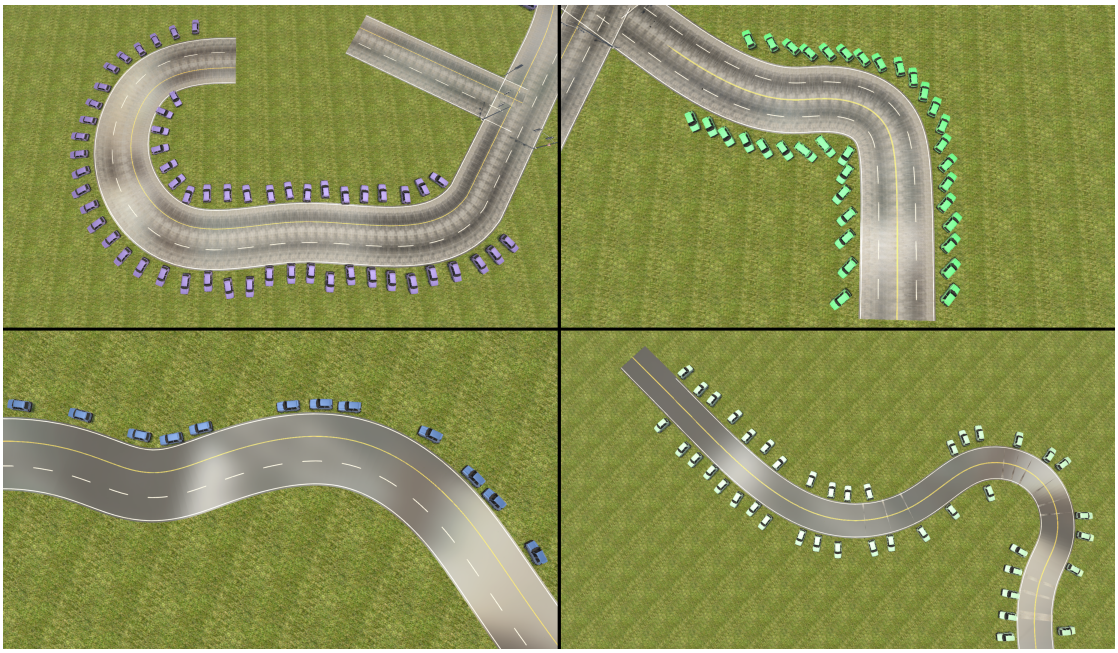

Figure 54: Four examples to place parked cars.

5 Related Work

Wu et al. [70] proposed a fuel-economy optimization system which predicts the optimal acceleration/deceleration value using sensor and environment variables in a Lagrange multiplier method. Even though the authors tested their approach with human drivers, they could achieve a reduction between 12-31%. According to the survey in [51], Toyota planned to control the acceleration power in three different driving modes. Kang et al. [45] proposed a sensing and control system suitable for every car. Their system accelerates/decelerates the car to reach the optimal value for fuel-efficiency. However, the authors exclude any possible obstacle on the road like pedestrians or other vehicles. Also, the testing environment is unknown and Kang et al. as well as other works just state "highway" or "urban environment".
Boriboonsomsin et al. evaluated how real-time fuel economy feedback affects driving behaviour and how much fuel can be saved [7]. Their results show that drivers could save up to 6% and the majority considers to improve their driving behaviour in the future. This implicates that people will support the fact that autonomous cars could drive as eco-friendly as possible. There are several websites which encourage fuel-efficient practices [18, 41] by giving suggestions or even instructions on how this could be accomplished. Similar sites list rules of how driving situations should be handled like left turns or crossing an intersection facing a stop sign [28, 41]. Converting this information into a set of rules, one can define how cars should drive in specific scenarios and penalize them when they are not.
Mersky et al. introduce a method to test the fuel economy of self-driving vehicles when following another car [50]. The authors estimate fuel consumption using the Virginia Tech Comprehensive Fuel Consumption model [57]. The authors tested various planning algorithms on different vehicles and could observe savings between -3% to 5%. Another possible way to reduce fuel consumption is by looking at gear-shifting strategies. The authors in [55] proposed several gear shifting methods based on dynamic programming and could reduce fuel consumption while at the same time maintain the driveability. Ganti et al. use GPS data to map the fuel economy on the streets [26]. That way, drivers can choose the most efficient way, but these routes can be long and take more time.
Other publications focus on modelling the fuel consumption of cars. One baseline method is the Virginia Tech Comprehensive Fuel Consumption model by Rakha et al. [57]. In contrast to prior work, the authors exclude the calibration of model parameters, which could only be done by in-laboratory or field data collections, and instead use publicly available vehicles and roadway parameters. Also, the authors overcome the issue of bang-bang control models, which can be seen as a threshold when some control system should be activated. The results show that their model highly correlates with in-field measurements.
Concerning traffic signs, Taal et al. [65] use procedural generation to place traffic signs in a given urban scenario by analysing the topological structure and road characteristics. Gambi et al. [25] use a procedural content generator with a Genetic Algorithm to evolve test cases which causes the car to drive off the road as many times as possible. Abdessalem et al. [6] detect feature interaction by using multi-objective search-based test generation. Stocco et al. [63] instead try to avoid potential safety-critical misbehaviours by measuring the confidence of a Deep Neural Network. Mustafa et al. [53] use procedural content generation to create urban layouts which reduce water depth during flood scenarios. To the best of my knowledge, there are no test generators which stress the maximum fuel-inefficiency of autonomous cars. Additionally, most works focus on passenger safety and feature interaction. My thesis is a contribution to the lacking field of testing fuel-efficiency of self-driving cars in simulations by using procedural content generation to create urban-like scenarios.

6 Conclusions

In this thesis, I present a novel approach for assessing the fuel-efficiency of self-driving cars and a test generator capable of creating simulations that feature elements of urban scenarios, like traffic lights, signs, parked cars, as well as a rich set of road and intersection configurations. By using a procedural content generator to generate virtual urban-like scenarios, autonomous cars can be scored in various urban environments, with features that stress fuel-inefficient driving behaviours. My extensive evaluation showed that my scoring function can spot fuel-inefficient driving behaviour and correlates strongly with the consumed fuel. The results also revealed that my test generator can create efficient test cases with traffic lights which show errors in traffic light detection systems.

7 Future Work

Despite the positive results I could achieve with my thesis, there is room for improvements. My ongoing work covers (i) improving the scoring function to increase the correlation with the consumed fuel even further; (ii) adding more fuel-inefficiency metrics to find even more weaknesses of self-driving cars; (iii) add more complex traffic like traffic jams, more vehicles, more variety to the routes that can be taken by existing participants, other models, different speed limits and driving behaviour; (iv) use more models for parked cars; (v) increase the realism of the environment by adding sidewalks, parking spots for cars, street light and possibly buildings; (vi) adding roundabouts which could replace intersections; (vii) improving the geometry of roads to be more versatile, e.g. replacing the mandatory straight segments with standard segments; (viii) adding more traffic rules to the test oracle like how to behave correctly at yield signs or the right-before-left rule; (ix) improving existing metrics for fuel-inefficiency like creating a engine fuel map for RPM validation; (x) using search-based testing techniques like evolutionary algorithms to improve test cases over time; (xi) using fog to evaluate the traffic light detection system to compare and interpret the results; (xii) overcome the current limitations of intersection design to generate richer variations; (xiii) give the user more power over traffic lights control.
I could also imagine training a reinforcement learning agent by using my scoring function. It would be interesting to see whether the agent will be able to drive fuel-efficient or not.

References

[1] Affinis AI Laboratory. Affinis ai laboratory. https://github.com/affinis-lab.

[2] Legal Headquarters Allgemeine Deutsche Automobil-Club e. V (ADAC). Parken: Neue regeln, alte strafen. https://www.adac.de/verkehr/recht/verkehrsvorschriften-deutschland/parken/, 2020.

[3] Mediatus Andreas 06. Zeichen 306 - vorfahrtstraße, stvo 1970.svg. https://de.wikipedia.org/wiki/Datei:Zeichen_306_-_Vorfahrtstra%C3%9Fe,_StVO_1970.svg. Last visited on 16.09.20.

[4] B25Mitch. Introduction to vehicle creation. https://wiki.beamng.com/Introduction_to_Vehicle_Creation.

[5] BeamNG GmbH. BeamNG.research. https://www.beamng.gmbh/research.

[6] R. Ben Abdessalem, A. Panichella, S. Nejati, L. C. Briand, and T. Stifter. Testing autonomous cars for feature interaction failures using many-objective search. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 143–154, 2018.

[7] Kanok Boriboonsomsin, Alexander Vu, and Matthew Barth. Eco-driving: Pilot evaluation of driving behavior changes among u.s. drivers. University of california transportation center, working papers, University of California Transportation Center, 2010.

[8] F. Bounini, D. Gingras, V. Lapointe, and H. Pollart. Autonomous vehicle and real time road lanes detection and tracking. In *2015 IEEE Vehicle Power and Propulsion Conference (VPPC)*, pages 1–6, 2015.

[9] Thibaud Briand and Pascal Monasse. Theory and practice of image b-spline interpolation. *Image Processing On Line*, 8:99–141, 2018.

[10] Stadtentwicklung Bundesministerium für Verkehr, Bau. Allgemeine verwaltungsvorschrift zur straßenverkehrs-ordnung (vwv-stvo). http://www.verwaltungsvorschriften-im-internet.de/bsvwvbund_26012001_S3236420014.htm.

[11] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2722–2730, 2015.

[12] K.C. Colwell. This is how your car's engine works. https://www.caranddriver.com/features/a26962316/how-a-car-works/.

[13] Christophe Croux and Catherine Dehon. Influence functions of the spearman and kendall correlation measures. *Statistical Methods & Applications*, 19(4):497–515, Nov 2010.

[14] Paul Dekraker, Daniel Barba, Andrew Moskalik, and Karla Butters. Constructing engine maps for full vehicle simulation modeling. In *WCX World Congress Experience*. SAE International, apr 2018.

[15] Denelson83. Stop sign.png. https://commons.wikimedia.org/wiki/File:Stop_sign.png. Last visited on 16.09.20.

[16] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[17] Eartheasy. Tips for fuel-efficient driving. `https://learn.eartheasy.com/guides/fuel-efficient-driving/`.

[18] ecodrivingusa.com. Ecodriving practices. `https://www.pepperdine.edu/sustainability/content/ecodrivingusa-eco-driving-practices.pdf`.

[19] U.S. EPA. Sources of greenhouse gas emissions. `https://www.epa.gov/ghgemissions/sources-greenhouse-gas-emissions`, 2017.

[20] estama. Accurate fuel consumption. `https://www.beamng.com/threads/accurate-fuel-consumption.16742/#post-262774`, 2013.

[21] Stuart Taylor (Chain Bear F1). What is engine mapping and how does it affect a car's behaviour? — plus mini q&a. `https://youtu.be/1cr4uIzkD0k`.

[22] filipbasara0. Carla traffic light dataset. `https://drive.google.com/drive/folders/1TXkPLWlNgauPhQnKEoPDZsx7Px1MD9n_`.

[23] filipbasara0. traffic-light-detection-module. `https://github.com/affinis-lab/traffic-light-detection-module`.

[24] Forschungsgesellschaft für Straßen-und Verkehrswesen. *Richtlinien für Lichtsignalanlagen : RiLSA : Lichtzeichenanlagen für den Straßenverkehr*. FGSV Verlag GmbH, 2015.

[25] A. Gambi, M. Mueller, and G. Fraser. Asfault: Testing self-driving car software using search-based procedural content generation. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 27–30, 2019.

[26] Raghu K. Ganti, Nam Pham, Hossein Ahmadi, Saurabh Nangia, and Tarek F. Abdelzaher. Greengps: A participatory sensing fuel-efficient maps application. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, page 151–164, New York, NY, USA, 2010. Association for Computing Machinery.

[27] Jordan Golson. Tesla driver killed in crash with autopilot active, nhtsa investigating. *The Verge*, 2016.

[28] NSW Government. Intersections and turning. `https://www.rms.nsw.gov.au/roads/safety-rules/road-rules/intersections.html`.

[29] Michael Heine. Crash. `https://youtu.be/kiD8AYsFKpI`.

[30] Michael Heine. Drive through red traffic light. `https://youtu.be/P3ckKR6dVgY`.

[31] Michael Heine. Drive through stop sign. `https://youtu.be/NBtvV9IrGbo`.

[32] Michael Heine. Stop at green traffic light. `https://youtu.be/t8OsyUgnUtE`.

[33] Michael Heine. Stop at priority sign. `https://youtu.be/6Mcu6wVkBlA`.

[34] Michael Heine. Succeeding test case 1. `https://youtu.be/2y6nYWOawx8`.

[35] Michael Heine. Succeeding test case 2. `https://youtu.be/9cJ5yX68FtQ`.

[36] Michael Heine. Traffic light modes and cycles. https://youtu.be/TvZF2vMQ6-c.

[37] Michael Heine. Traffic video 1. https://youtu.be/RSk3Vost15k.

[38] Michael Heine. Traffic video 2. https://youtu.be/1oURP4ls5b0.

[39] Michael Heine. Traffic video 3. https://youtu.be/vp10T44xASs.

[40] Michael Heine. Traffic video 4. https://youtu.be/hn-E9AwmJ2o.

[41] https://www.theorieexamen.nl. Car theory. https://www.theorieexamen.nl/auto-theorie/.

[42] https://www.theorieexamen.nl. Car theory ecodriving. https://www.theorieexamen.nl/auto-theorie/bewust_rijden.

[43] https://www.theorieexamen.nl. Dutch traffic signs. https://www.theorieexamen.nl/verkeersborden-overzicht/verkeersborden_b.

[44] S. Jaafari and Kourosh Heidari Shirazi. A comparison on optimal torque vectoring strategies in overall performance enhancement of a passenger car. *Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics*, 230, 01 2016.

[45] Lei Kang, Bozhao Qi, Dan Janecek, and Suman Banerjee. Ecodrive: A mobile sensing and control system for fuel efficient driving. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, MobiCom '15, page 358–371, New York, NY, USA, 2015. Association for Computing Machinery.

[46] Lewis Kingston. What is an electronic control unit? ph explains. https://www.pistonheads.com/news/features/what-is-an-electronic-control-unit-ph-explains/37771.

[47] Vladlen Koltun. Vladlen koltun: Autonomous driving: The way forward (july 2020). https://youtu.be/XmtTjqimW3g?t=3169.

[48] A.M. Liaquat, H.H. Masjuki, M.A. Kalam, I.M. Rizwanul Fattah, M.A. Hazrat, M. Varman, M. Mofijur, and M. Shahabuddin. Effect of coconut biodiesel blended fuels on engine performance and emission characteristics. *Procedia Engineering*, 56:583 – 590, 2013. 5th BSME International Conference on Thermal Engineering.

[49] Phil McCausland. Self-driving uber car that hit and killed woman did not recognize that pedestrians jaywalk. *NBC News*, 2019.

[50] Avi Chaim Mersky and Constantine Samaras. Fuel economy testing of autonomous vehicles. *Transportation Research Part C: Emerging Technologies*, 65:31 – 48, 2016.

[51] Alexander Meschtscherjakov, David Wilfinger, Thomas Scherndl, and Manfred Tscheligi. Acceptance of future persuasive in-car interfaces towards a more economic driving behaviour. In *Proceedings of the 1st International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI '09, page 81–88, New York, NY, USA, 2009. Association for Computing Machinery.

[52] AAMCO Minnesota. What makes a car go when you step on the gas. https://aamcominnesota.com/what-makes-a-car-go-when-you-step-on-the-gas/.

[53] Ahmed Mustafa, Xiao Wei Zhang, Daniel G Aliaga, Martin Bruwier, Gen Nishida, Benjamin Dewals, Sébastian Erpicum, Pierre Archambeau, Michel Pirotton, and Jacques Teller. Procedural generation of flood-sensitive urban layouts. *Environment and Planning B: Urban Analytics and City Science*, 47(5):889–911, 2020.

[54] United Nations. Vienna convention road sign b1-v1.svg. `https://commons.wikimedia.org/wiki/File:Vienna_Convention_road_sign_B1-V1.svg`. Last visited on 16.09.20.

[55] Viet Dac Ngo, Jose A. Colin Navarrete, Theo Hofman, Maarten Steinbuch, and Alex Serrarens. Optimal gear shift strategies for fuel economy and driveability. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 227(10):1398–1413, 2013.

[56] U.S. Department of Energy. How do gasoline cars work? `https://afdc.energy.gov/vehicles/how-do-gasoline-cars-work`.

[57] Hesham A. Rakha, Kyoungho Ahn, Kevin Moran, Bart Saerens, and Eric Van den Bulck. Virginia tech comprehensive power-based fuel consumption model: Model development and testing. *Transportation Research Part D: Transport and Environment*, 16(7):492 – 503, 2011.

[58] RedBolide. '87 vw golf mk2 gti [update 10/17]. `https://www.beamng.com/threads/87-vw-golf-mk2-gti-update-10-17.4922/`.

[59] Morgan Stanley Research. Autonomous Cars. Self-Driving the New Auto Industry Paradigm, 2013.

[60] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 102–118, Cham, 2016. Springer International Publishing.

[61] Noor Shaker, Julian Togelius, and Mark J Nelson. *Procedural content generation in games*. Springer, 2016.

[62] Anthony Stark. Brake specific fuel consumption (bsfc). `https://x-engineer.org/automotive-engineering/internal-combustion-engines/performance/brake-specific-fuel-consumption-bsfc/`. [Online; accessed 22-September-2020].

[63] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. Misbehaviour prediction for autonomous driving systems, 2019.

[64] Straßenverkehrsordnung (StVO). § 8 vorfahrt. `https://www.stvo.de/strassenverkehrsordnung/96-8-vorfahrt`.

[65] Fieke Taal and Rafael Bidarra. Procedural generation of traffic signs. In *Proceedings of the Eurographics Workshop on Urban Data Modelling and Visualisation*, pages 17–23. Eurographics Association, 2016.

[66] CARLA Challenge Team. Carla challenge. `https://carlachallenge.org/challenge/`.

[67] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat,

Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[68] Eric W. Weisstein. B-spline. `https://mathworld.wolfram.com/B-Spline.html`. Last visited on 11.09.20.

[69] Wikibooks. Introduction to numerical methods/interpolation — wikibooks, the free textbook project. `https://en.wikibooks.org/w/index.php?title=Introduction_to_Numerical_Methods/Interpolation&oldid=3573289`, 2019. [Online; accessed 16-September-2020].

[70] Changxu Wu, Guozhen Zhao, and Bo Ou. A fuel economy optimization system with applications in vehicles with human drivers and autonomous vehicles. *Transportation Research Part D: Transport and Environment*, 16(7):515 – 524, 2011.

8 Appendix

8.1 XML Files Example

```
<?xml version='1.0' encoding='utf-8'?>
<criteria>
    <environment>urban0.dbe.xml</environment>
    <author>Michael Heine</author>
    <participants>
        <participant color="White" id="ego" model="ETK800">
            <initialState x="1.0" y="-2.5" orientation="0" />
            <movement>
                <waypoint x="80.0000000000" y="-2.5000000000" tolerance="2" lane="0" speed="13.8" />
                <waypoint x="85.0000000000" y="-2.5000000000" tolerance="2" lane="0" speed="0" />
                <waypoint x="190.0000000000" y="-2.5000000000" tolerance="2" lane="1" speed="13.8" />
                <waypoint x="195.0000000000" y="-2.5000000000" tolerance="2" lane="1" speed="0" />
            </movement>
        </participant>
    </participants>
    <success>
        <scPosition participant="ego" x="195" y="-2.5" tolerance="3" />
    </success>
    <failure>
        <scDamage participant="ego" />
        <scLane participant="ego" onLane="offroad" />
    </failure>
    <name>Fuel Efficiency Test</name>
    <triggerPoints>
        <triggerPoint action="stop" x="60" y="-2.5" triggeredBy="ego" duration="7" tolerance="2" />
        <triggerPoint action="switchLights" initState="green" switchTo="red" tolerance="2"
                      triggeredBy="ego" triggers="traffic_light_manual_0" x="60" y="-2.5" />
    </triggerPoints>
</criteria>
```

Figure 55: A XML criteria file.

Figures 55 and 56 show the XML files of an example test case.

8.2 Traffic Light Experiment Test Cases

In this section, I will show all eleven test cases that were used to test the traffic light detection system. The brightness of the background reflects the time of day used in the test case. Each figure also contains the spawn point of the ego car, the roads, the success point and the positions of traffic lights and signs. The test cases were generated randomly and can be seen in Figure 57 and 58.

```xml
<?xml version='1.0' encoding='utf-8'?>
<environment>
    <author>Michael Heine</author>
    <timeOfDay>0</timeOfDay>
    <lanes>
        <lane leftLanes="1" markings="true" rightLanes="1">
            <laneSegment x="0.0000000000" y="0.0000000000" width="10" />
            <laneSegment x="200.0000000000" y="0.0000000000" width="10" />
        </lane>
        <lane leftLanes="1" markings="true" rightLanes="1">
            <laneSegment x="100.0000000000" y="-100.0000000000" width="10" />
            <laneSegment x="100.0000000000" y="100.0000000000" width="10" />
        </lane>
    </lanes>
    <obstacles>
        <trafficlightsingle x="95" y="-5" zRot="0" mode="manual" sign="yield" oid="traffic_light_manual_0"
                            facingEgo="True" />
        <trafficlightsingle x="105" y="-5" zRot="90" mode="off" sign="priority" />
        <trafficlightsingle x="95" y="5" zRot="270" mode="off" sign="priority" />
        <trafficlightsingle x="105" y="5" zRot="180" mode="off" sign="yield" />
        <parkedCar x="10" y="-7" zRot="0" />
        <parkedCar x="14" y="-7" zRot="0" />
        <parkedCar x="18" y="7" zRot="0" />
    </obstacles>
</environment>
```

Figure 56: A XML environment file.

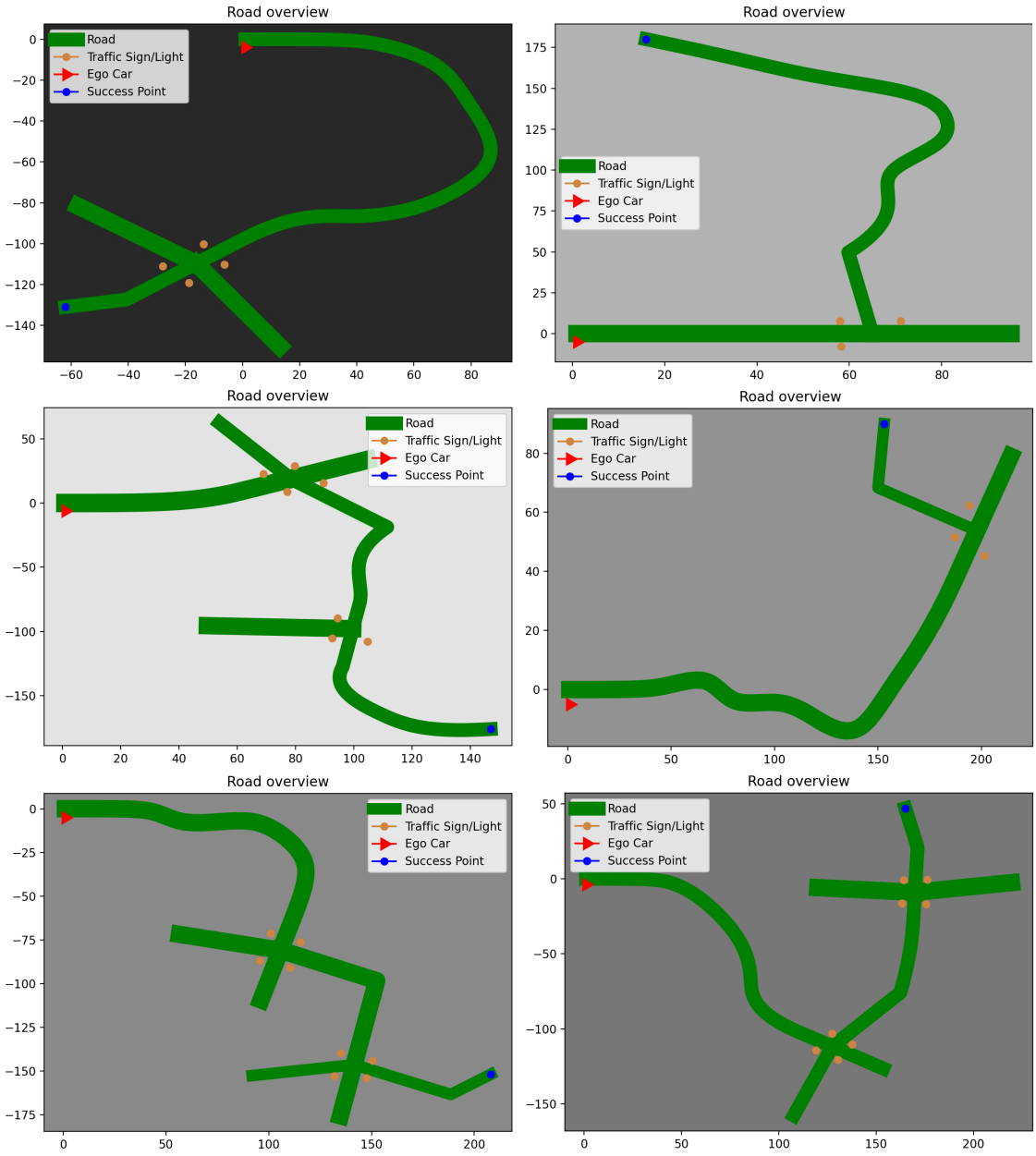

Figure 57: The first six out of eleven test cases of the traffic lights experiment.

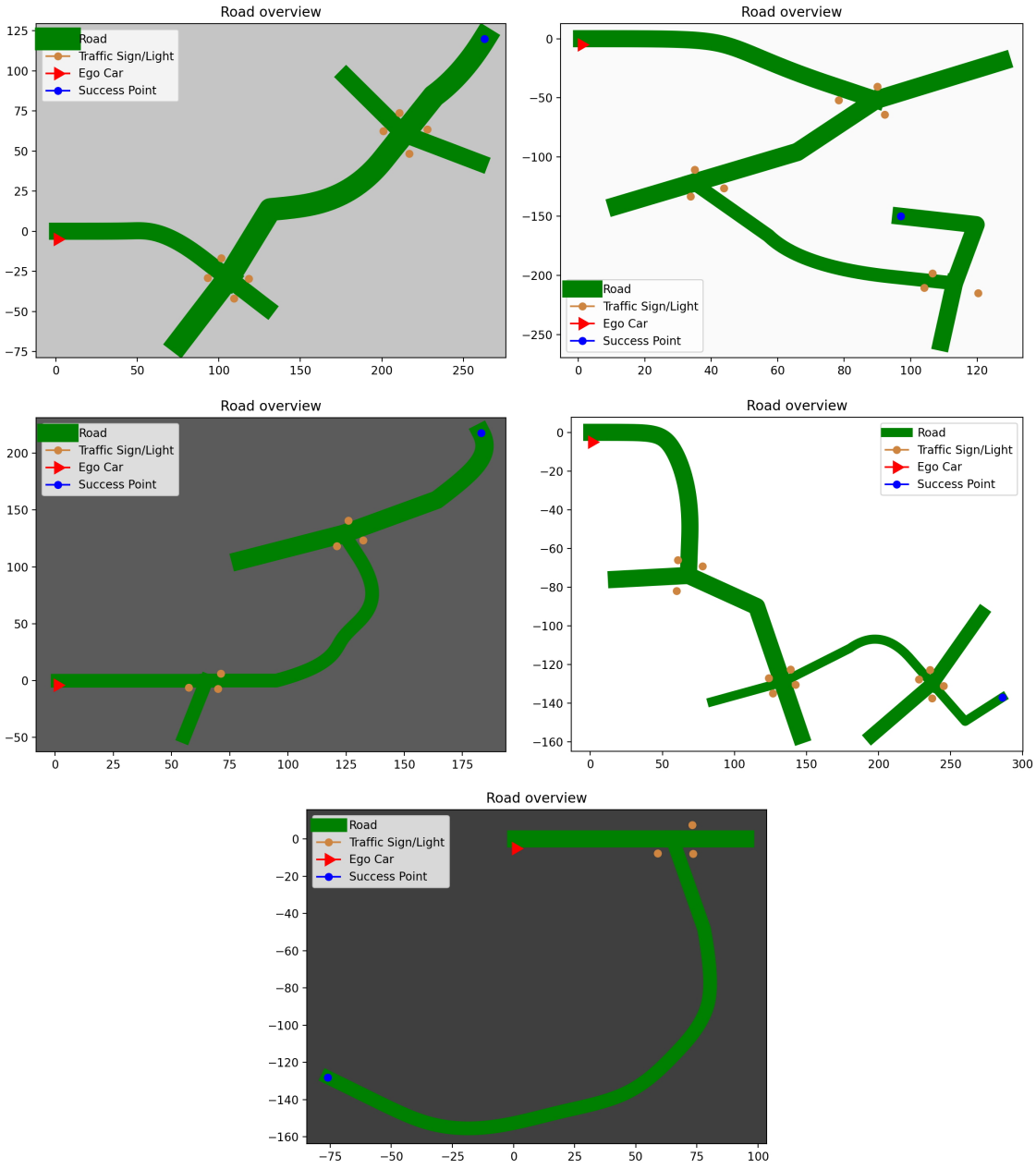

Figure 58: The last five out of eleven test cases of the traffic lights experiment.

Eigenständigkeitserklärung

Hiermit bestätige ich _____(Name), dass ich die vorliegende Arbeit selbstständig und ohne unzulässige Hilfe verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die wörtlich und sinngemäß übernommenen Passagen aus anderen Werken kenntlich gemacht habe. Die Arbeit ist weder von mir noch von einer anderen Person an der Universität Passau oder an einer anderen Hochschule zur Erlangung eines akademischen Grades bereits eingereicht worden.

_____ _____

Ort, Datum Unterschrift